CARLETON UNIVERSITY

cuPID

Requirements Analysis Document

Team [Code First, Think Later]
Kevin Hua
Hendrik Knoetze
Juhandré Knoetze

Submitted to:
Dr. Christine Laurendeau
COMP 3004: Object Oriented Software Engineering
School of Computer Science
Carleton University

October 14, 2015

Contents

1 Introduction **4**
1.1 Purpose of System 4
1.2 Overview of Document 4

2 Proposed System **5**
2.1 Overview 5
2.2 Functional Requirements 5
2.3 Non-Functional Requirements 6
2.4 System Models 8
2.4.1 Use Case Model 8
2.4.2 Object Model 26
2.4.3 Dynamic Model 42

Figures

1 High-level Use Case Diagram 9
2 Detailed Use Case Diagram - Administrator 10
3 Detailed Use Case Diagram - Student 11
4 Detailed Use Case Diagram - User Input 12
5 Detailed Use Case Diagram - Common Errors 13
6 Entity Objects Diagram 26
7 Control/Boundary Objects for ManageEnrollment 28
8 Control/Boundary Objects for EditProfile 28
9 Control/Boundary Objects for ManageProject 28
10 Control/Boundary Objects for CreateNewProject 29
11 Control/Boundary Objects for LaunchPPID 29
12 Control/Boundary Objects for InsufficientStudentsError 29
13 Control/Boundary Objects for ViewPPIDResults 30
14 Control/Boundary Objects for ViewPPIDSummary 30
15 Control/Boundary Objects for ViewPPIDDetails 30
16 Control/Boundary Objects for EditProject 31
17 Control/Boundary Objects for EditProjectDetails 31
18 Control/Boundary Objects for JoinProject 31
19 Control/Boundary Objects for LeaveProject 32
20 Control/Boundary Objects for EditPersonalValues 32
21 Control/Boundary Objects for EditDesiredValues 32
22 Control/Boundary Objects for InvalidInputError 33
23 Control/Boundary Objects for EditProjectName 33
24 Control/Boundary Objects for ProjectExistsError 33
25 Control/Boundary Objects for EditGroupSize 34
26 Control/Boundary Objects for InvalidGroupSizeError 34
27 Control/Boundary Objects for StorageError 34
28 Control/Boundary Objects for StorageReadError 35
29 Control/Boundary Objects for StorageOpenError 35
30 Control/Boundary Objects for StorageWriteError 35

31 SM-01 - State Machine for User/Student/Administrator Login . . . 42
32 SM-02 - State Machine for Student Join Project . . . 43
33 SM-03 - State Machine for Student Leave Project . . . 43
34 SM-04 - State Machine for Profile Details . . . 43
35 SM-05 - State Machine for Project Student Enrollment . . . 43
36 SM-06 - State Machine for Project Name . . . 44
37 SM-07 - State Machine for Project PPID Results . . . 44
38 SM-08 - State Machine for Group Enrollment . . . 44
39 SQ-01 - Sequence Diagram for ManageEnrollment . . . 45
40 SQ-02 - Sequence Diagram for EditProfile . . . 46
41 SQ-03 - Sequence Diagram for ManageProject . . . 46
42 SQ-04 - Sequence Diagram for CreateNewProject . . . 47
43 SQ-05 - Sequence Diagram for LaunchPPID . . . 47
44 SQ-06 - Sequence Diagram for InsufficientStudentsError . . . 48
45 SQ-07 - Sequence Diagram for ViewPPIDResults . . . 48
46 SQ-08 - Sequence Diagram for ViewPPIDSummary . . . 49
47 SQ-09 - Sequence Diagram for ViewPPIDDetails . . . 49
48 SQ-10 - Sequence Diagram for EditProject . . . 50
49 SQ-11 - Sequence Diagram for EditProjectDetails . . . 50
50 SQ-12 - Sequence Diagram for JoinProject . . . 51
51 SQ-13 - Sequence Diagram for LeaveProject . . . 51
52 SQ-14 - Sequence Diagram for EditPersonalValues . . . 52
53 SQ-15 - Sequence Diagram for EditDesiredValues . . . 52
54 SQ-16 - Sequence Diagram for InvalidInputError . . . 53
55 SQ-17 - Sequence Diagram for EditProjectName . . . 53
56 SQ-18 - Sequence Diagram for ProjectExistsError . . . 54
57 SQ-19 - Sequence Diagram for EditGroupSize . . . 54
58 SQ-20 - Sequence Diagram for InvalidGroupSizeError . . . 55
59 SQ-21 - Sequence Diagram for StorageError . . . 55
60 SQ-22 - Sequence Diagram for StorageReadError . . . 56
61 SQ-23 - Sequence Diagram for StorageOpenError . . . 56
62 SQ-24 - Sequence Diagram for StorageWriteError . . . 57

Tables

1 Functional Requirements . . . 6
2 Non-Functional Requirements . . . 7
3 High-Level Use Case Descriptions . . . 10
4 Detailed Use Case Descriptions - Administrator . . . 11
5 Detailed Use Case Descriptions - Student . . . 12
6 Detailed Use Case Descriptions - User Input . . . 13
7 Detailed Use Case Descriptions - Common Errors . . . 14
8 Entity Object Data Dictionary . . . 27
9 Control Object Data Dictionary . . . 36
10 Boundary Object Data Dictionary . . . 40

1 Introduction

– project –

[cuPID]

1.1 Purpose of System

Team projects are typically assigned in University courses in order to develop and foster good teamwork related skills, which are crucial in most future endeavours, notably prospective employment. Unfortunately, the task of separating students into balanced and compatible teams has always been a nigh impossible feat - hardly a year passes that doesn't boast at least a single team brimming with contention. There just seems to be too many nuances that are involved in building a perfect team for professors to account for, often leading to random or pseudo-random assignment of teams. Another possibility that professors might employ would be to allow the students themselves to form their teams. Regrettably, this option also leads to much strife - students tend to select friends or acquaintances as partners. While this solution might seem good, it is sadly the case that good friends often make bad project partners. There is also the case that many students have not made friends or acquaintances yet that they could ask to be their partners, thus leaving them to form a team with others in their situation.

Both current options leave much to be desired. Our firm, [Code First, Think Later], has been hired to design a system that would provide a better solution to this long-standing issue. Thus, the purpose of our system is to separate students into teams with others of similar personality and skills, eliminating the frequent torment associated with ill-matched teams.

1.2 Overview of Document

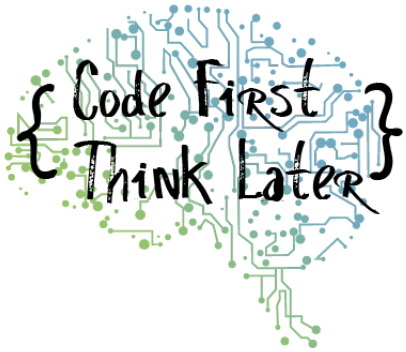
This report provides an overview of the design decisions we made for the cuPID project, including use cases, object models, and dynamic models. The goal of this document is to clearly impart our vision of this program to Dr. Christine Laurendeau, our esteemed client. In furtherance to this goal, we have painstakingly organized our report to maximize both basic legibility and traceability. Before delving into the more complex details of our proposed system, we have included a general overview of the elements we wish to incorporate, as well as some details with regards to our unique and innovative algorithm.

Following the short synopsis, you will find, the requested requirements, cleanly organized into separate functional and non-functional tables. Note that each requirement is assigned a unique but informative identifier - all future references to this particular requirement will cite its corresponding code. The purpose of the system is, again, to ensure that the reading of this document is as simple and pleasant an experience as possible.

Directly after the requirements, we have the various system models. We have chosen to separate them into three primary categories: the use cases, the objects, and the dynamic models. The goal of this separation scheme is to stagger the introduction of different components such that the reader does not feel bombarded. Thus, you will first see the various possible scenarios and how the system will flow with regards to these scenarios. Subsequently, the various objects will be introduced and described in detail. Finally, we will delve into the more detailed sequence diagrams that will build

on information from the prior sections. This section will handle such things as timings and the flows for the various possible states of the system.

Best regards,



(Team [Code First, Think Later])

2 Proposed System

2.1 Overview

The system that we are proposing to build offers a simplistic and clean interface designed for the single purpose of generating well-rounded and personality-compatible teams. To this end, we will allow two possible users: the Administrator, typically the professor, and the Student. We want to really make a clear distinction between these two users - all that the Student user can do is access their personal profile and control their participation to projects, while the Administration is in charge of all other aspects of project maintenance. Solely the Administrator will be able to run and view the results of our specialized Partner Matching Algorithm. This algorithm, broadly speaking, will be analyzing four aspects of a person in order to generate the best possible teams: programming or other relevant skills, leadership skills, personality, and availability.

In this section, we will showcase a comprehensive breakdown of all the various components of our proposed system. For the sake of legibility, we have decided to organize the content in such a way that we move naturally from simplest to most complex - in other words, each subsequent section builds off the previous.

2.2 Functional Requirements

Functional requirements are the result of us carefully analyzing the problem statement that you have submitted and extracting all the key desired features. In other words, we have converted your request into a list of requirements that we will fulfill. For example, you mention that it is essential that this software can create projects. All the main features that you have expressed the desire to have have been matched with the appropriate user and included in the following table, **Table 1**.

Table 1: Functional Requirements

ID	Description
F-01	Students must be able to add themselves to any number of projects.
F-02	Students must be able to remove themselves from a project they are currently part of.
F-03	Students must be able to edit their Project Partner Profile.
F-03-01	Students must be able to modify the values representing their own qualifications.
F-03-02	Students must be able to modify the values representing what qualifications they desire in potential partners (what they prioritize, for example).
F-04	Students must be able to view their Project Partner Profile.
F-05	Administrators must be able to create new projects.
F-06	Administrators must be able to edit existing projects.
F-06-01	Administrators must be able to modify the team size parameter for individual projects.
F-06-02	Administrators must be able to add a student to a selected project.
F-06-03	Administrators must be able to remove a student from a selected project.
F-06-04	Administrators must be able to modify the project name parameter for individual projects.
F-07	Administrators must be able to launch the PPID for a specific project.
F-08	Administrators must be able to view the summary results of the last-run PPID.
F-09	Administrators must be able to view the detailed results of the last-run PPID.

2.3 Non-Functional Requirements

Non-functional requirements are differentiable from the functional requirements in that they are not the requested features of the project, but rather the various standards our proposed system must meet. For example, you expressed the desire that the system be designed specifically for use with the Linux Ubuntu platform, and that all code must be written in the C++ programming language. Also included in **Table 2** are assorted promises to meet various standards that you have not included in your problem statement but that either we believe should be required, or is required by law.

Table 2: Non-Functional Requirements

ID	Category	Description
NF-01	*Usability*	The option to customize the key bindings for default keyboard short cuts must be available for all users.
NF-02	*Usability*	The option to change the interface colour scheme to accomodate for colour-blindness must be available for all users.
NF-03	*Usability*	The size of the 'buttons' on the interface must be large enough to accomodate for users that want to use a touch-screen monitor.
NF-04	*Reliability*	The system will not terminate due to errors 95% of the time.
NF-05	*Reliability*	Saved data will remain uncorrupted at least 95% of the time.
NF-06	*Reliability*	Saved information must be backed up.
NF-07	*Implementation*	The system must be written in the C++ language.
NF-08	*Implementation*	The system must use the Qt framework for creating the graphical user interface (G.U.I.).
NF-09	*Implementation*	The system must run in the Ubuntu 14.04 virtual machine.
NF-10	*Performance*	The PPID will take no longer than 5 seconds to finish running.
NF-11	*Performance*	The system will not take longer than 2 seconds to perform any save operation.
NF-12	*Performance*	The system will not take longer than 2 seconds to perform any load operation.
NF-13	*Supportability*	The system must be extensible to a client/server architecture.
NF-14	*Supportability*	The system must be portable to Windows and Mac operating systems.
NF-15	*Supportability*	Later system versions will support the saved data of at least the last two versions.
NF-16	*Legal*	Users must accept a Terms of Service agreement in order to use the software.
NF-17	*Legal*	The system will be released under the GPL (GNU General Public License).
NF-18	*Legal*	The software will not install any supplementary software onto the users computers without the users expressed consent (section 8 of CASL (Canada's Anti-Spam Legislation)).

ID	Category	Description
NF-19	*Interface*	The software must provide an API to cuLearn.
NF-20	*Interface*	The software must provide an API to Wix, the website builder.
NF-21	*Interface*	The software must provide an API to Basecamp, the Business management system.
NF-22	*Operation*	A forum should be provided for users to report any bugs they may encounter.
NF-23	*Operation*	There will be a call center available for any enquiries.
NF-24	*Operation*	A live-chat system with certified users will be available during standard operating hours.
NF-25	*Packaging*	The software will be available to download as a standalone executable.
NF-26	*Packaging*	The client is in charge of installing it on their own machine.
NF-27	*Packaging*	Installation instructions will be included.

2.4 System Models

System models are some great tools that we have decided to incorporate into our system proposition to better convey the various functionalities and interactions in our system. They provide a strong visual representation of the various ideas that we have, including general scenarios (use-cases) such as "Administrator creates a new project," interactions between different objects, and precise sequence diagrams that display what each object does and for approximately how long. Specifically, you will be provided with use case diagrams with corresponding descriptions, entity, control, and boundary objects with corresponding dictionaries, state machine diagrams, and sequence diagrams.

Use case diagrams and descriptions are useful for showcasing all the possible actions we want to incorporate into our system, with the added bonus of clearly identifying the requirements each satisfies. Entity objects delve deeper and represent all the users and other objects in the game, listing all the attributes and associations that each has. The purpose of including this model is to exhibit all the potential behavior and aspects each object has access to. The next model, state machines, takes a large step backwards; here we see a more abstracted view of the system - specifically, a diagram showing the possible states that each control object can experience. In other words, this model does not see the system as a whole, but rather focuses on a very minute aspect of the system and explains how it works. Finally. sequence diagrams expand on the previously mentioned use cases and transforms the detailed descriptions into visual representations that also display the proportion of time spent at each step. Furthermore, the creation and destruction of objects is also displayed in this diagram. While it can be argued that each model exhibits varying levels of informativeness, each is required to fully understand the next.

2.4.1 Use Case Model

Use case models are pairings of diagrams that depicts the actions that a specified user can take in a tree-like format with detailed descriptions of each action. The actions are divided into two three types: the high-level cases, denoted with «initiates», the detailed cases, denoted with «include»,

and the error cases, denoted with «extend». To augment readability, we have decided to have a separate section for the high-level cases and grouping the detailed and error cases together. One concession we did make was to separate the different actions by *User*, or actor.

The format of this section will be separated in two parts. For the first part, we will further subdivide into two parts: the high-level use cases and the detailed use cases. These two sub-sections will be identically formatted insofar as the diagrams will precede the more detailed descriptions, which are organized in a clean table format. Following this first part, the second part will contain the more detailed flow of events for every use case in the previous part (both high-level and detailed).

High-level Use Cases

This section contains the high-level use cases - essentially, the most abstracted action each user (actor) can take. The diagram below, **Figure 1**, illustrates the interaction between users, defined by the stick figure, and the system, defined by the rectangular box. The ovals represent the highest level actions that can be taken, with a connecting line to the user that can initiate them. Specifically, we have the *Student User* who can initiate the high-level action of *ParticipateInProjects*, and we have the *Administrator User* who can initiate in the action of *ManageProject*. These actions are purposefully vague and general because they are the "parent action" that includes several other more specific actions.

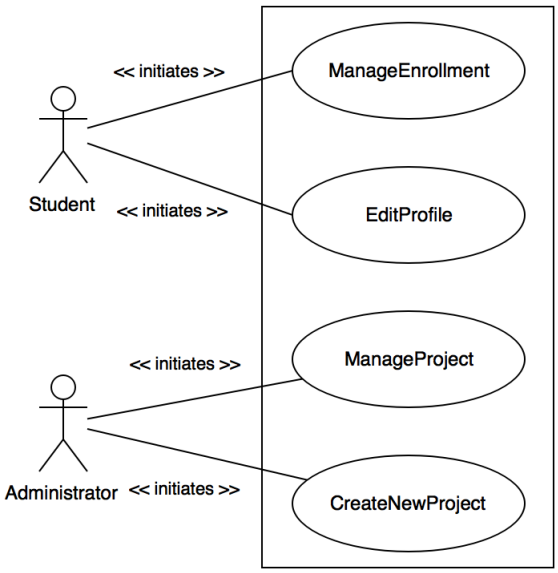


Figure 1: High-level Use Case Diagram

Pulling the use cases from the diagram, we have next a list of the use cases in table format, **Table 3**, with a short and succinct descriptions beside. The purpose of this table, on top of having the diagram, is for ease of access with regards to information. While it is very nice to see a diagram, nothing quite beats the traditional table when it comes to laying out data. Another benefit is, like in the case of *EditProfile*, it shows what detailed use cases are included in these higher-level cases.

Table 3: High-Level Use Case Descriptions

ID	Name	Description
UC-01	**ManageEnrollment**	The Student accesses and modifies the projects the Student is enrolled in.
UC-02	**EditProfile**	The Student modifies the parameters of their profile (includes EditPersonalValues and EditDesiredValues).
UC-03	**ManageProject**	The Administrator manages projects as well as having the option to create a new instance of a project (includes manipulating student participation and access to the PPID and its results).
UC-04	**CreateNewProject**	The Administrator creates a new instance of a project.

Detailed Use Cases

This section contains the more detailed use cases - the specific cases, as it were, of the the much more general high-level use cases. Whereas the high-level cases help give a good idea of how things are situated, detailed use cases are much more informative because they encompass all the finer details of the proposed software. Sure, it's nice to know that an *Administrator* can *ManageProject*, but I am sure that you would agree with us that this is very vague. What exactly is "Managing Projects"? Well, that is the puprose of this section.

The below figure and table, **Figure 2** and **Table 4**, cover the use cases that concern the *Administrator*.

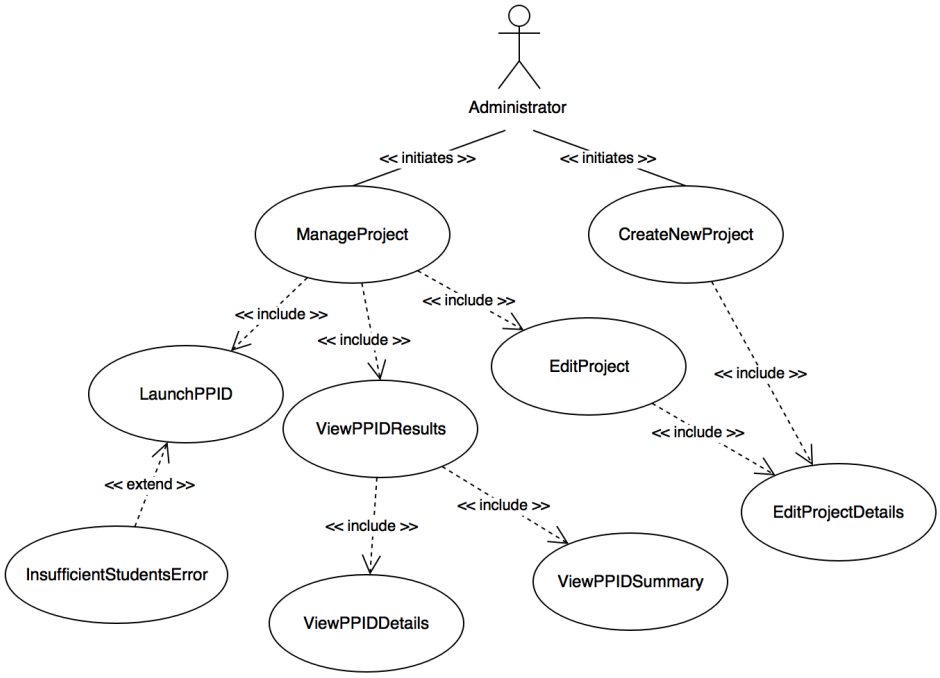


Figure 2: Detailed Use Case Diagram - Administrator

The next table, **Table 4**, contains all the information about each particular use case for the

Administrator, though forgoing the actual relationships. A unique identifier for each use case is also assigned here.

Table 4: Detailed Use Case Descriptions - Administrator

ID	Name	Description
UC-05	**LaunchPPID**	The Administrator runs the PPID for a specified project.
UC-06	**InsufficientStudentsError**	The system reports that there are not enough Students registered in the project to run the PPID.
UC-07	**ViewPPIDResults**	The Administrator views the results of the PPID for a project.
UC-08	**ViewPPIDSummary**	The Administrator views a summary of the groups that are created.
UC-09	**ViewPPIDDetails**	The Administrator views a detailed analysis of the grouping decisions.
UC-10	**EditProject**	The Administrator modifies an aspect of the project (includes Student participation and project parameters).
UC-11	**EditProjectDetails**	The Administrator modifies the parameters of a specified project.

The following figure, **Figure 3**, outlines the use cases and their relationships with each other for the *Student User*. Note here that the *Student* has far fewer possible use cases as compared to the *Administrator* - there are only two major things that *Students* can do: edit their profile, and manage their enrollment in projects. They have no access to finer details such as project settings.

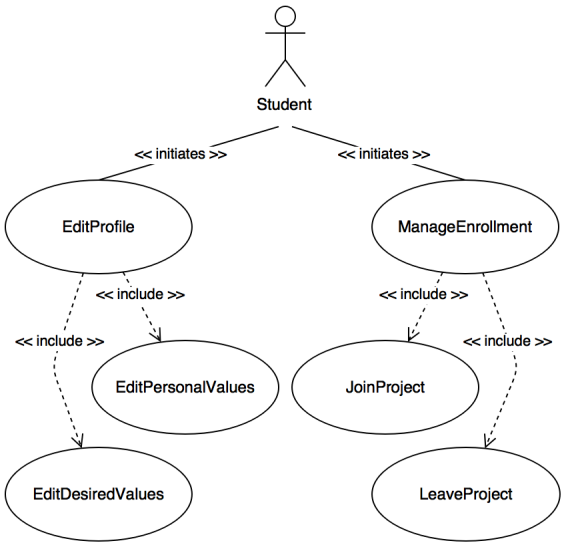


Figure 3: Detailed Use Case Diagram - Student

The ensuing table, **Table 5** describes each of the possible use cases that a *Student User* can use/encounter. Please also note that we have decided to not include any error cases in this particular diagram to maximize legibility.

Table 5: Detailed Use Case Descriptions - Student

ID	Name	Description
UC-12	**JoinProject**	The student adds themselves to an existing project.
UC-13	**LeaveProject**	The student removes themselves from an existing project.
UC-14	**EditPersonalValues**	The student modifies the personal details in his profile (what they have to offer).
UC-15	**EditDesiredValues**	The student modifies the desired details in his profile (what they are looking for in potential team members).

The following diagram, **Figure 4**, displays the possible ways each user can input into the system and their associated errors. Note that for most of the cases, the inputs are value-based and thus all refer to the same *InvalidInputError* use case.

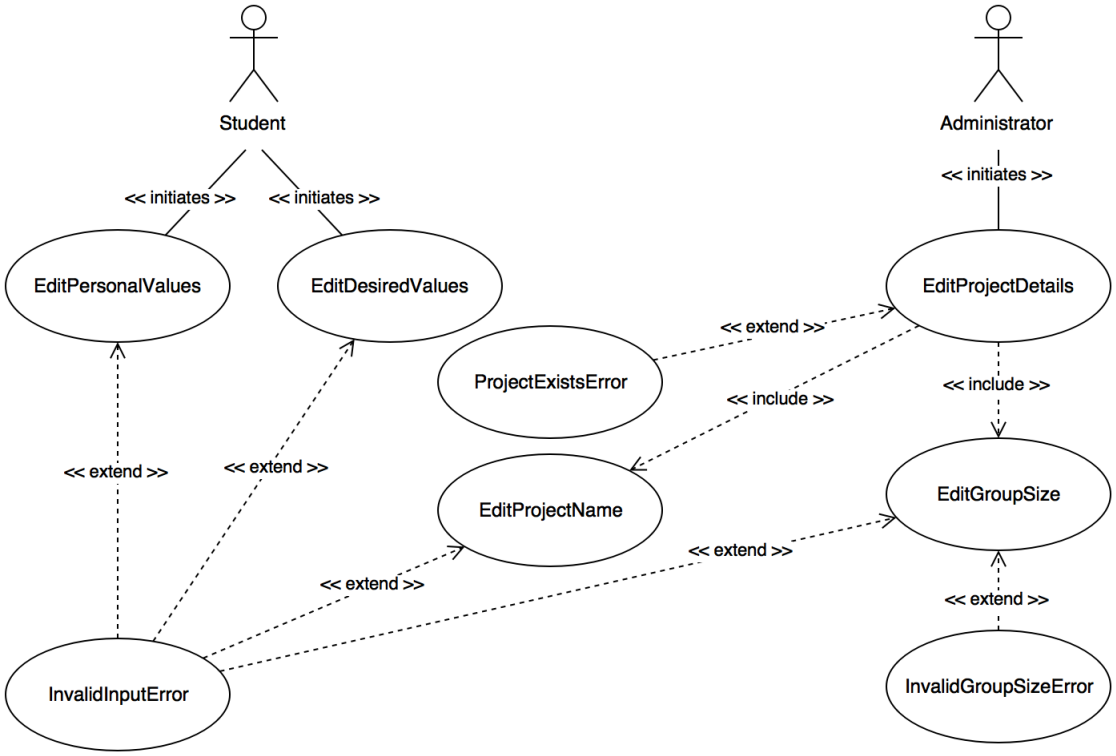


Figure 4: Detailed Use Case Diagram - User Input

Table 6, the following table, pulls the use cases from **Figure 4** and provides both a detailed description and a unique identifier for each. That said, please note that several of the cases present in **Figure 4** are absent in this table - *EditPersonalValues*, *EditDesiredValues*, and *EditProjectDetails* all have already been described in **Table 4** and **Table 5**.

Table 6: Detailed Use Case Descriptions - User Input

ID	Name	Description
UC-16	**InvalidInputError**	The system reports that the given value is invalid.
UC-17	**EditProjectName**	The Administrator sets the name of the project.
UC-18	**ProjectExistsError**	The system reports that a project with the given name already exists.
UC-19	**EditGroupSize**	The Administrator sets the minimum and maximum group sizes to be used by the PPID.
UC-20	**InvalidGroupSizeError**	The system reports that the given group size is illegal.

The following diagram, **Figure 5**, shows which use cases, for both the *Administrator* and the *Student* Users, can trigger the *StorageError* use case. In other words, which of the priorly mentioned use cases interact with reading files, writing files, and opening the Storage.

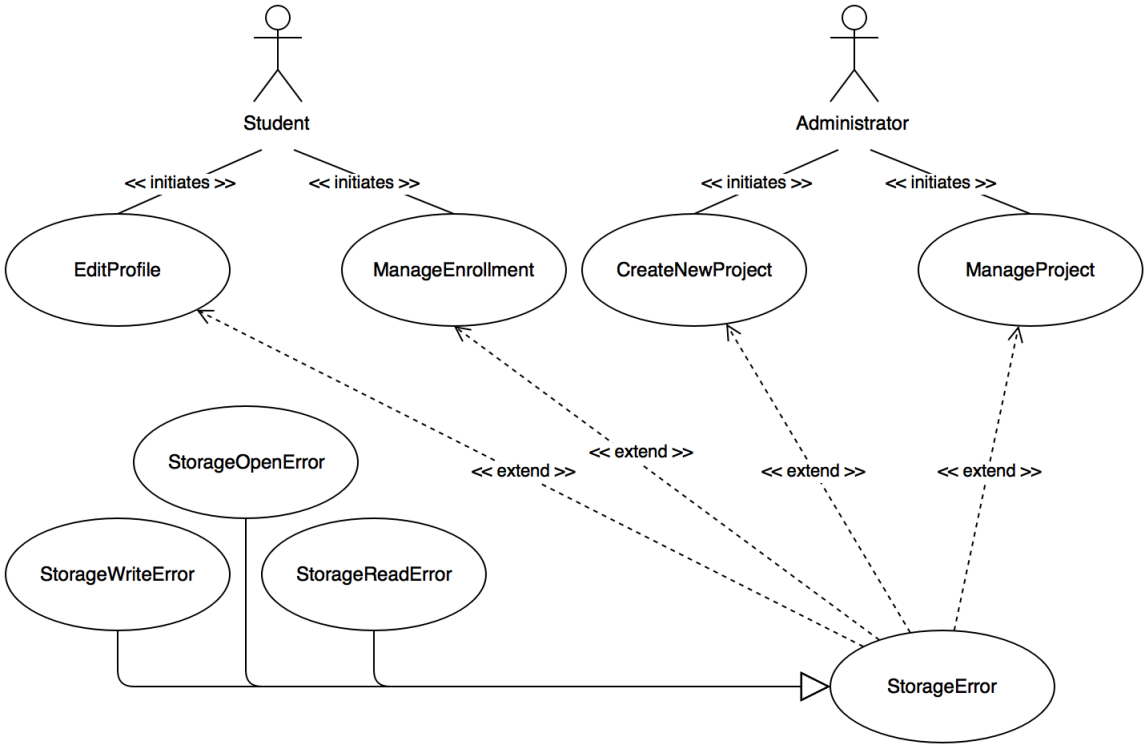


Figure 5: Detailed Use Case Diagram - Common Errors

The table below, **Table 7**, covers all the the descriptions and assignment of unique identifiers for the storage-related error use cases. Note that the first use case in this table, *StorageError*, is a generalization of the other three use cases.

Table 7: Detailed Use Case Descriptions - Common Errors

ID	Name	Description
UC-21	**StorageError**	The system reports that there is an error with the stored information for cuPID (generalizes StorageReadError, StorageOpenError, and StorageWriteError).
UC-22	**StorageReadError**	The system reports that an error occurred while attempting to read from the Storage.
UC-23	**StorageOpenError**	The system reports that an error occurred while attempting to open the Storage.
UC-24	**StorageWriteError**	The system reports that an error occurred while attempting to write to the Storage.

Table-Based Use Case Flow of Events

In the following section, we will delve into each of the priorly mentioned use cases in much more detail. Specifically, participating actors, detailed flow of events, entry/exit conditions, quality requirements, if applicable, and references to which requirements each address shall all be provided for each use case. To ensure ease of reading, we will maintain the exact same ordering as listed in the above section.

The *ManageEnrollment* (**UC-01**) allows Student actors to both join or leave existing projects. It is a high-level use case that includes both the *JoinProject* and the *LeaveProject* use cases.

Use Case Identifier	**UC-01**
Name	**ManageEnrollment**
Participating Actors	Initiated by Student
Flow of Events	1. The Student chooses the manage enrollment option. 2. The system displays a list of projects. 3. The Student chooses a project. 4. The system displays a menu with two options. The Student can choose to join the project (include use case **JoinProject**), or leave the project (include use case **LeaveProject**).
Entry Conditions	• The User is operating cuPID as a Student.
Exit Conditions	
Quality Requirements	
Traceability	F-01, F-02

The *EditProfile* (**UC-02**) allows Student actors to edit their profiles. Specifically, this is a high-level use case that includes both *EditPersonalValues* and *EditDesiredValues* use cases.

Use Case Identifier	**UC-02**
Name	**EditProfile**
Participating Actors	Initiated by Student
Flow of Events	1. The Student chooses the edit profile option. 2. The system displays a menu with two options. The Student can choose to edit their own values for the attributes (include use case **EditPersonalValues**) or edit their desired partner values for the attributes (include use case **EditDesiredValues**).
Entry Conditions	• The User is operating cuPID as a Student.
Exit Conditions	
Quality Requirements	
Traceability	F-03, F-03-01, F-03-02

The *ManageProject* (**UC-03**) allows Administrator actors to manage project details. This is a high-level use case that includes *LaunchPPID*, *ViewPPIDResults*, *EditProject*, and *DeleteProject*.

Use Case Identifier	**UC-03**
Name	**ManageProject**
Participating Actors	Initiated by Administrator
Flow of Events	1. The Administrator selects the manage project option. 2. The system displays a list of projects. 3. The Administrator chooses a project. 4. The system displays a menu with three options. The Administrator can choose to launch the PPID (include use case **LaunchPPID**), view PPID results (include use case **ViewPPIDResults**), edit an existing project (include use case **EditProject**).
Entry Conditions	• The User is operating cuPID as an Administrator.
Exit Conditions	
Quality Requirements	
Traceability	F-05, F-06, F-07, F-08, F-09

The *CreateNewProject* (**UC-04**) allows Administrator actors to create a new project. This is a high-level use case that includes the *EditProjectDetails* use case.

Use Case Identifier	**UC-04**
Name	**CreateNewProject**
Participating Actors	Initiated by Administrator
Flow of Events	1. The Administrator selects the create new project option. 2. The Administrator enters in the initial project values (include use case **EditProjectDetails**). 3. The system displays the new project to the Administrator.
Entry Conditions	• The User is operating cuPID as an Administrator. • The Administrator has chosen to create a new project.
Exit Conditions	• The project is created.
Quality Requirements	
Traceability	F-05

The *LaunchPPID* (**UC-05**) allows Administrator actors to launch the sorting algorithm on a specified project and save the results.

Use Case Identifier	**UC-05**
Name	**LaunchPPID**
Participating Actors	Initiated by Administrator
Flow of Events	1. The system computes the PPID results. 2. The system saves the PPID results. 3. The system notifies the Administrator that the results have been computed.
Entry Conditions	• The User is operating cuPID as an Administrator. • The Administrator has chosen to run the PPID.
Exit Conditions	• The PPID results are saved. • The PPID results are displayed to the Administrator.
Quality Requirements	• The PPID will take no longer than 5 seconds to complete.
Traceability	F-07, NF-05

The *InsufficientStudentsError* (**UC-06**) allows the system to notify the Administrator actor if the Administrator actor attempts to run the PPID when there are insufficient students in the project.

Use Case Identifier	**UC-06**
Name	**InsufficientStudentsError**
Participating Actors	Communicates with Administrator
Flow of Events	1. The system informs the Administrator that there isn't enough Students in the Project to launch the PPID. 2. The launch of the PPID is stopped.
Entry Conditions	• This use case **extends** the LaunchPPID use case. It is initiated by the Administrator whenever the Administrator launches the PPID on a project but there are not enough students to satisfy the minimum group size.
Exit Conditions	• The launch of the PPID is stopped.
Quality Requirements	
Traceability	NF-04

The *ViewPPIDResults* (**UC-07**) allows the Administrator actor to view the results of the last run PPID for the specified project. Includes the *ViewPPIDSummary* and *ViewPPIDDetails* cases.

Use Case Identifier	**UC-07**
Name	**ViewPPIDResults**
Participating Actors	Initiated by Administrator
Flow of Events	1. The system displays a menu with two options. The Administrator can choose to view summary PPID results (include use case **ViewPPIDSummary**) or view detailed PPID results (include use case **ViewPPIDDetails**).
Entry Conditions	• The User is operating cuPID as an Administrator. • The Administrator has chosen to view the PPID results, or the Administrator has finished launching the PPID.
Exit Conditions	
Quality Requirements	
Traceability	F-08, F-09

The *ViewPPIDSummary* (**UC-08**) allows Administrator actors to see the print out of the final decision of the PPID. In other words, it allows the Administrator to see the different groups that the PPID came up with.

Use Case Identifier	**UC-08**
Name	**ViewPPIDSummary**
Participating Actors	Initiated by Administrator
Flow of Events	1. The system displays a print out of each group, including the students in each group.
Entry Conditions	• The User is operating cuPID as an Administrator. • The Administrator has chosen to view the PPID summary.
Exit Conditions	
Quality Requirements	
Traceability	F-08

The *ViewPPIDDetails* (**UC-09**) allows Administrator actors to see the actual steps and reasoning of the PPID with regards to the last run PPID. This print out does include the decision (actual groups it decided on) as well as the inner-details.

Use Case Identifier	**UC-09**
Name	**ViewPPIDDetails**
Participating Actors	Initiated by Administrator
Flow of Events	1. The system displays the reasoning behind the computed groupings.
Entry Conditions	• The User is operating cuPID as an Administrator. • The Administrator has chosen to view the PPID grouping details.
Exit Conditions	
Quality Requirements	
Traceability	F-09

The *EditProject* (**UC-10**) allows Administrator actors to edit project details. Includes the *EditProjectDetails* use case.

Use Case Identifier	**UC-10**
Name	**EditProject**
Participating Actors	Initiated by Administrator
Flow of Events	1. The Administrator enters in new project values (include use case **EditProjectDetails**). 2. The system displays the updated project to the Administrator.
Entry Conditions	• The Administrator has chosen to edit a Project.
Exit Conditions	• The project is updated.
Quality Requirements	
Traceability	F-06

The *EditProjectDetails* (**UC-11**) allows Administrator actors to modify project details like the possible group sizes and project name. Includes the *EditGroupSize* and *EditProjectName* use cases.

Use Case Identifier	**UC-11**
Name	**EditProjectDetails**
Participating Actors	Initiated by Administrator
Flow of Events	1. The system displays a menu with two options. The Administrator can choose to edit the minimum and maximum group size (include use case **EditGroupSize**), or the name of the Project (include use case **EditProjectName**). 2. The system saves the project details.
Entry Conditions	• The User is operating cuPID as an Administrator. • The Administrator has chosen to edit existing, or set the initial, project details.
Exit Conditions	• The project details have been saved.
Quality Requirements	
Traceability	F-05, F-06, F-06-01, F-06-04

The *JoinProject* (**UC-12**) allows Student actors to add themselves to an existing project that they are not already part of.

Use Case Identifier	**UC-12**
Name	**JoinProject**
Participating Actors	Initiated by Student
Flow of Events	1. The system adds the Student to the selected project. 2. The system notifies the Student that they have been added to the project.
Entry Conditions	• The User is operating cuPID as a Student. • The Student has selected to join a project.
Exit Conditions	• The addition of the Student to a project has been saved.
Quality Requirements	
Traceability	F-01

The *LeaveProject* (**UC-13**) allows Students actors to remove themselves from an existing project that they are a part of.

Use Case Identifier	**UC-13**
Name	**LeaveProject**
Participating Actors	Initiated by Student
Flow of Events	1. The Student selects a project from a list of their joined projects. 2. The system removes the Student from the selected project. 3. The system notifies the Student that they have been removed from the project.
Entry Conditions	• The User is operating cuPID as a Student. • The Student has chosen to remove themselves from a project.
Exit Conditions	• The removal of the Student from a project has been saved.
Quality Requirements	
Traceability	F-02

The *EditPersonalValues* (**UC-14**) allows Students to modify the values for the section pertaining to their personal details.

Use Case Identifier	**UC-14**
Name	**EditPersonalValues**
Participating Actors	Initiated by Student
Flow of Events	1. The system displays a menu of the Student's personal values that they can edit. 2. The Student edits their personal values. 3. The system saves the Student's personal values.
Entry Conditions	• The User is operating cuPID as a Student. • The Student has chosen to edit their personal values.
Exit Conditions	• The updated personal values have been saved.
Quality Requirements	
Traceability	F-03-01

The *EditDesiredValues* (**UC-15**) allows Student actors to modify the values for the section pertaining to the values they desire in teammates.

Use Case Identifier	**UC-15**
Name	**EditDesiredValues**
Participating Actors	Initiated by Student
Flow of Events	1. The system displays a menu of the Student's desired values that they can edit. 2. The Student edits their desired values. 3. The system saves the Student's desired values.
Entry Conditions	• The User is operating cuPID as a Student. • The Student has chosen to edit their desired values.
Exit Conditions	• The updated desired values have been saved.
Quality Requirements	
Traceability	F-03-02

The *InvalidInputError* (**UC-16**) allows the system to notify the Administrator or Student actors when they input incorrect values.

Use Case Identifier	**UC-16**
Name	**InvalidInputError**
Participating Actors	Communicates with Administrator, Student
Flow of Events	1. The system informs the Administrator/Student that the entered input is invalid. 2. Reverts the input to its original value.
Entry Conditions	• This use case **extends** EditPersonalValues, EditDesiredValues, EditProjectName, and EditGroupSize. This is initiated by the Administrator/Student when the Administrator/Student inputs invalid data.
Exit Conditions	• The invalid input is removed, and the input is reverted to its original value.
Quality Requirements	
Traceability	NF-04

The *EditProjectName* (**UC-17**) allows Administrator actors to modify the name of the current project.

Use Case Identifier	**UC-17**
Name	**EditProjectName**
Participating Actors	Initiated by Administrator
Flow of Events	1. The Administrator inputs a unique project name. 2. The system updates the project name. 3. The system notifies the Administrator that the name has updated.
Entry Conditions	• The User is operating cuPID as an Administrator. • The Administrator has chosen to set the project name.
Exit Conditions	• The project name is updated.
Quality Requirements	
Traceability	F-06-04

The *ProjectExistsError* (**UC-18**) allows the system to notify the Administrator actor if the desired project name is already in use.

Use Case Identifier	**UC-18**
Name	**ProjectExistsError**
Participating Actors	Communicates with Administrator
Flow of Events	1. The system informs the Administrator that the project name is already in use. 2. The saving of an updated project is cancelled.
Entry Conditions	• This **extends** the EditProjectDetails use case. It is initiated by the Administrator when the attempts to save a project that has the same project name as an existing project.
Exit Conditions	• The project update is cancelled and the project name is reverted to its original value.
Quality Requirements	
Traceability	NF-04

The *EditGroupSize* (**UC-19**) allows Administrator actors to modify the range of possible group sizes, which includes the maximum and minimum values.

Use Case Identifier	**UC-19**
Name	**EditGroupSize**
Participating Actors	Initiated by Administrator
Flow of Events	1. The Administrator inputs a new value for the minimum or maximum group size. 2. The system updates the project's minimum and maximum group size values. 3. The system notifies the Administrator that the group size has updated.
Entry Conditions	• The User is operating cuPID as an Administrator. • The Administrator has chosen to update the group size.
Exit Conditions	• The group size is updated.
Quality Requirements	
Traceability	F-06-01

The *InvalidGroupSizeError* (**UC-20**) allows the system to notify the Administrator actor that the desired group size is invalid.

Use Case Identifier	**UC-20**
Name	**InvalidGroupSizeError**
Participating Actors	Communicates with Administrator
Flow of Events	1. The system informs the Administrator that the entered group size is invalid. 2. The system reverts the group size to its original value.
Entry Conditions	• This use case **extends** EditGroupSize use case. It is initiated by the Administrator when the administrator sets the minimum group size higher than the maximum group size or vice versa.
Exit Conditions	• The group size is reverted.
Quality Requirements	
Traceability	NF-04

The *StorageError* (**UC-21**) allows the system to notify the Administrator and Student actors that an error has occured while trying to interact with the Storage.

Use Case Identifier	**UC-21**
Name	**StorageError**
Participating Actors	Communicates with Administrator, Student
Flow of Events	1. The system notifies the Administrator/Student that an error has occurred while interacting with the storage.
Entry Conditions	• This use case **extends** ManageProject, ManageEnrollment, EditProfile, and CreateNewProject use cases. An operation involving the storage has failed.
Exit Conditions	• The storage operation is cancelled.
Quality Requirements	
Traceability	NF-04

The *StorageReadError* (**UC-22**) allows the system to notify the Administrator and Student actors when an error has occured while trying to read from the storage.

Use Case Identifier	**UC-22**
Name	**StorageReadError**
Participating Actors	**Inherited** from StorageError use case.
Flow of Events	1. The system notifies the Administrator/Student that an error has occurred while reading from the storage.
Entry Conditions	• An attempt to read from the storage has failed.
Exit Conditions	• The read operation is cancelled.
Quality Requirements	
Traceability	NF-04

The *StorageOpenError* (**UC-23**) allows the system to notify the Administrator and Student actors when an error has occured while trying to opening the storage.

Use Case Identifier	**UC-23**
Name	**StorageOpenError**
Participating Actors	**Inherited** from StorageError use case.
Flow of Events	1. The system notifies the Administrator/Student that an error has occurred while opening the storage.
Entry Conditions	• An attempt to open the storage has failed.
Exit Conditions	• The open operation is cancelled.
Quality Requirements	
Traceability	NF-04

The *StorageWriteError* (**UC-24**) allows the system to notify the Administrator and Student actors when an error has occured while trying to save to the storage.

Use Case Identifier	**UC-24**
Name	**StorageWriteError**
Participating Actors	**Inherited** from StorageError use case.
Flow of Events	1. The system notifies the Administrator/Student that an error has occurred while saving to the storage.
Entry Conditions	• An attempt to save to the storage failed.
Exit Conditions	• The save is cancelled.
Quality Requirements	
Traceability	NF-04

2.4.2 Object Model

Object models can be separated into three sections: the entity objects, the control objects, and the boundary objects. As a whole, object models serve to show the high-level concepts of the system, the different kinds of persistent information, and their relationships/properties. In essence, we model how different objects interact with one another, either calling each other or simply getting called to display some information.

Entity Objects

Entity objects are objects that contain persistent information that other objects can access or modify. The following diagram, **Figure 6**, is an Entity Object Diagram that showcases the different relationships between the entities of this software. For example, Student and Administrator both inherit from User, whereas Students 'have-a' profile.

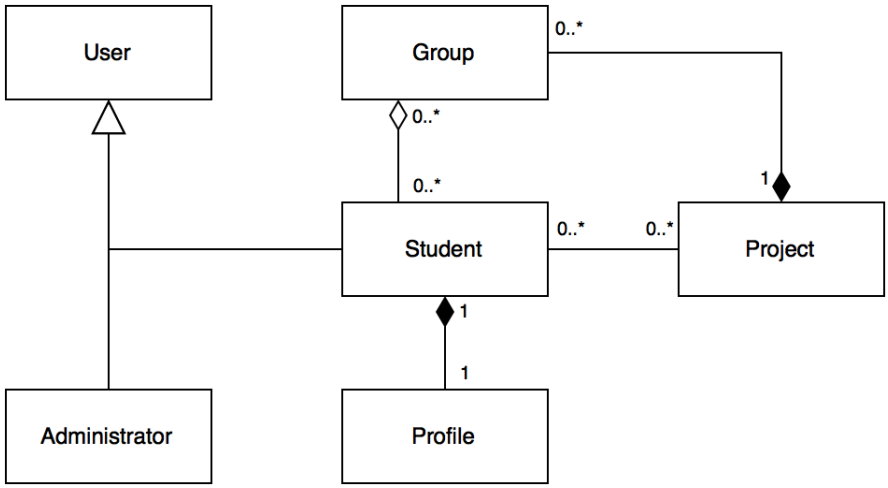


Figure 6: Entity Objects Diagram

The following table, **Table 8**, is the Data Dictionary for the Entity Objects. Here, we list out all the attributes and associations for each of the entity objects, as well as provide a short but detailed definition of each.

Table 8: Entity Object Data Dictionary

ID	Entity Object	Attributes/ Associations	Definition
CL-01	**User**	• name • id	A **User** that has the ability to access the cuPID system. Applies to **UC-16**, **UC-21**, **UC-22**, **UC-23**, **UC-24**
CL-02	**Student**	• studentNumber • profile • projects	A **Student** is a specific type of User that can create/edit their Profiles and join/leave existing Projects. Applies to **UC-01**, **UC-02**, **UC-12**, **UC-13**, **UC-14**, **UC-15**
CL-03	**Administrator**		An **Administrator** is a specific type of user that can create/edit projects and run/view results of the PPID. Applies to **UC-03**, **UC-04**, **UC-05**, **UC-07**, **UC-08**, **UC-09**, **UC-11**, **UC-10**, **UC-06**, **UC-17**, **UC-18**, **UC-19**, **UC-20**
CL-04	**Group**	• members	A **Group** is a collection of Student Users that are participating in a specified Project. Every Project contains at least 1 Group, and the Students in a Group are decided when the Administrator runs the PPID algorithm. Applies to **UC-05**, **UC-07**, **UC-08**, **UC-09**
CL-05	**Project**	• name • students • groupSize	A **Project** contains a collection of Students that will get sorted into Groups. These are made and edited by the **Administrator**. Applies to **UC-01**, **UC-03**, **UC-04**, **UC-05**, **UC-07**, **UC-08**, **UC-09**, **UC-11**, **UC-10**, **UC-12**, **UC-13**, **UC-17**, **UC-18**, **UC-19**, **UC-20**
CL-06	**Profile**	• details	A **Profile** is a set of responses maintained by each Student. These values are the necessary input for the PPID algorithm to run. Applies to **UC-02**, **UC-14**, **UC-15**

Control and Boundary Objects

Control objects are a special kind of object whose purpose is to fulfill use cases - the control objects are the objects that are doing all the tying together. They use function calls to access information from entity objects and create boundary objects to notify/interact with the actors (users). Boundary objects are used to handle the interaction between actors and the system. For example, if something erroneous has occurred, it is the boundary objects that notify the user.

Control and boundary objects for ManageEnrollment (UC-01):

The following figure, **Figure 7**, handles the interactions between the control object, *ManageEnrollmentControl*, and the corresponding boundary objects, *ProjectListNotice, EnrollmentOption*, and *ProjectSelectionOption*. The control object here manages **UC-01**, ManageEnrollment.

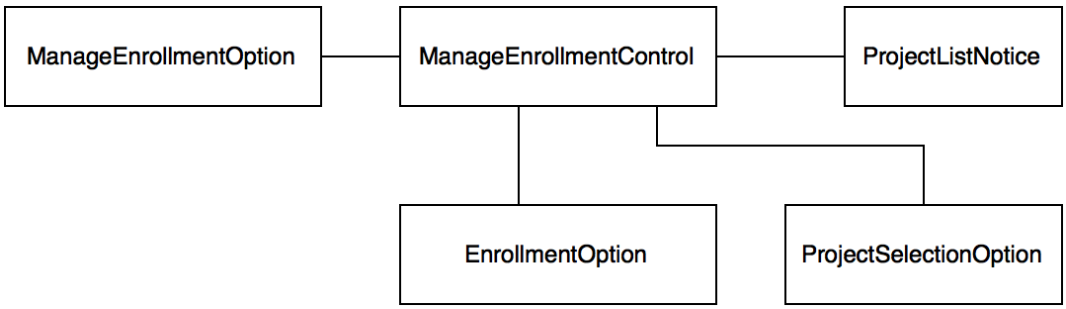


Figure 7: Control/Boundary Objects for ManageEnrollment

Control and boundary objects for EditProfile (UC-02):

The following figure, **Figure 8**, handles the interactions between the control object, *EditProfileControl*, and the corresponding boundary object, *ProfileObject*. The control object here manages **UC-02**, EditProfile.

Figure 8: Control/Boundary Objects for EditProfile

Control and boundary objects for ManageProject (UC-03):

The following figure, **Figure 9**, handles the interactions between the control object, *ManageProjectControl*, and the corresponding boundary objects, *ManageOption, ProjectListNotice,* and *ProjectSelectionOption*. The control object here manages **UC-03**, ManageProject.

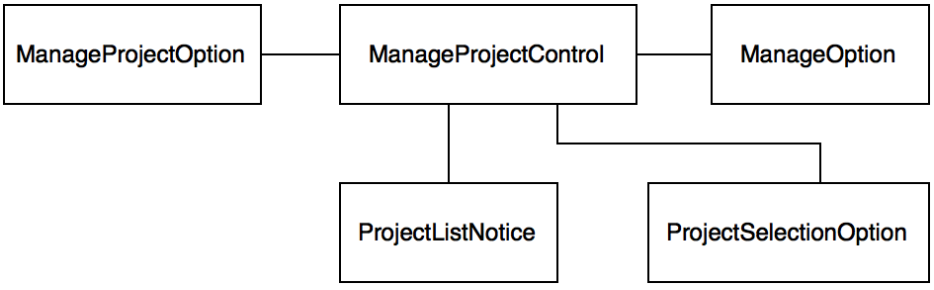


Figure 9: Control/Boundary Objects for ManageProject

Control and boundary objects for CreateNewProject (UC-04):

The following figure, **Figure 10**, handles the interactions between the control object, *CreateNewProjectControl*, and the corresponding boundary objects, *ProjectDetailsForm* and *ProjectCreateNotice*. The control object here manages **UC-04**, CreateNewProject.

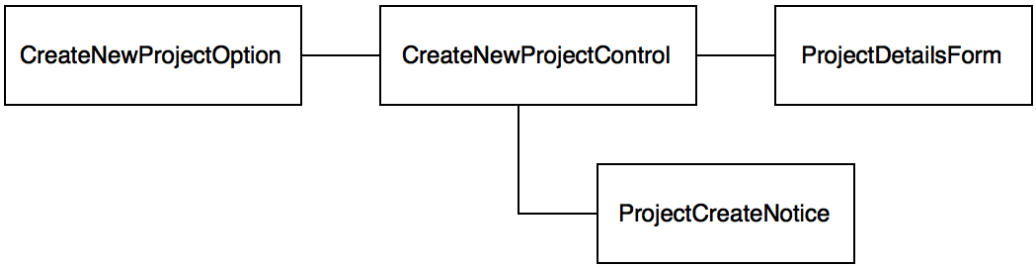


Figure 10: Control/Boundary Objects for CreateNewProject

Control and boundary objects for LaunchPPID (UC-05):

The following figure, **Figure 11**, handles the interactions between the control object, *LaunchPPIDControl*, the corresponding boundary object, *PPIDCompleteNotice*, and the entity object, *Project*. The control object here manages **UC-05**, LaunchPPID.

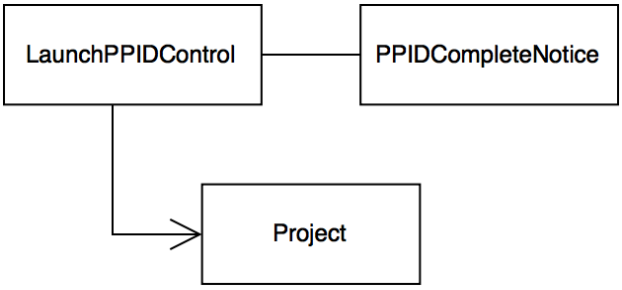


Figure 11: Control/Boundary Objects for LaunchPPID

Control and boundary objects for InsufficientStudentsError (UC-06):

The following figure, **Figure 12**, handles the interactions between the control objects, *LaunchPPIDControl* and *InsufficientStudentsErrorControl*, and the corresponding boundary object, *InsufficientStudentErrorNotice*. The control object here manages **UC-06**, InsufficientStudentsError.

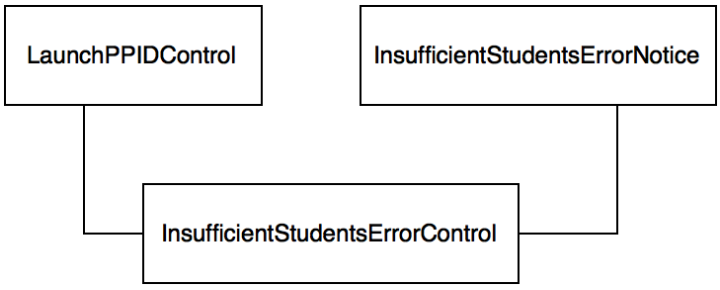


Figure 12: Control/Boundary Objects for InsufficientStudentsError

Control and boundary objects for ViewPPIDResults (UC-07):

The following figure, **Figure 13**, handles the interactions between the control object, *ViewPPIDResultsControl* and the corresponding boundary object, *ResultsOption*. The control object here manages **UC-07**, ViewPPIDResults.

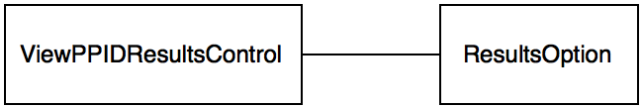


Figure 13: Control/Boundary Objects for ViewPPIDResults

Control and boundary objects for ViewPPIDSummary (UC-08):

The following figure, **Figure 14**, handles the interactions between the control object, *ViewPPIDSummaryControl*, the corresponding boundary object, *SummaryResultsNotice*, and the entity object, *Project*. The control object here manages **UC-08**, ViewPPIDSummary.

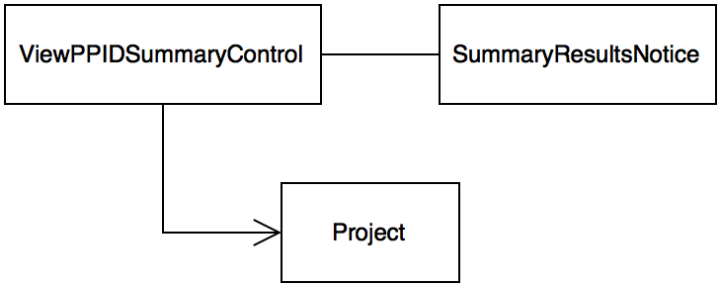


Figure 14: Control/Boundary Objects for ViewPPIDSummary

Control and boundary objects for ViewPPIDDetails (UC-09):

The following figure, **Figure 15**, handles the interactions between the control object, *ViewPPIDDetailsControl*, the corresponding boundary object, *DetailsResultsNotice*, and the entity object, *Project*. The control object here manages **UC-09**, ViewPPIDDetails.

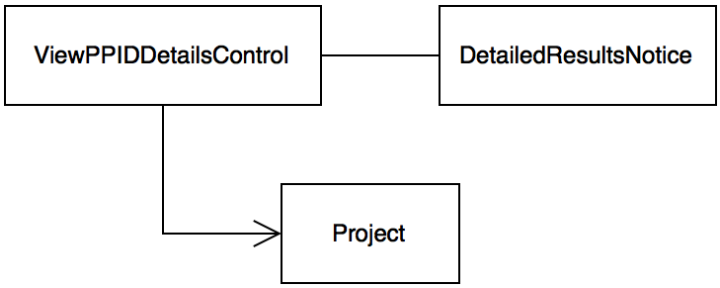


Figure 15: Control/Boundary Objects for ViewPPIDDetails

Control and boundary objects for EditProject (UC-10):

The following figure, **Figure 16**, handles the interactions between the control object, *EditProjectControl* and the corresponding boundary objects, *ProjectDetailsForm* and *DetailsUpdateNotice*. The control object here manages **UC-10**, EditProject.

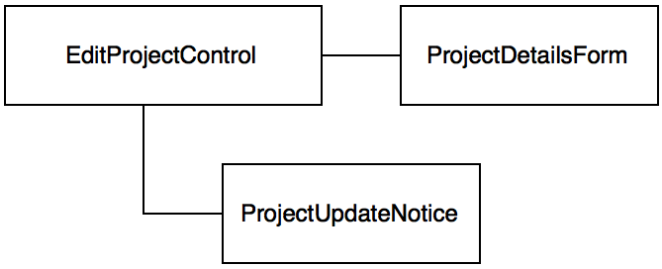


Figure 16: Control/Boundary Objects for EditProject

Control and boundary objects for EditProjectDetails (UC-11):

The following figure, **Figure 17**, handles the interactions between the control object, *EditProjectDetailsControl*, the corresponding boundary objects, *ProjectDetailsForm* and *DetailsUpdateNotice*, and the entity object, *Project*. The control object here manages **UC-11**, EditProjectDetails.

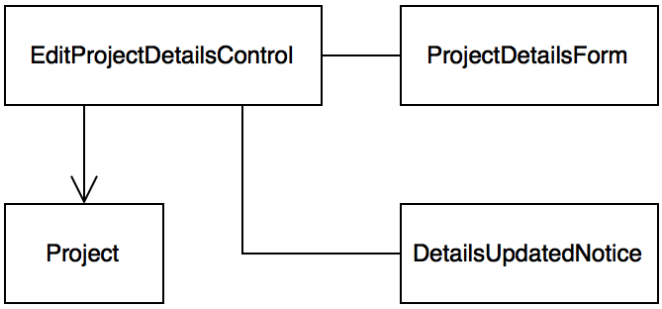


Figure 17: Control/Boundary Objects for EditProjectDetails

Control and boundary objects for JoinProject (UC-12):

The following figure, **Figure 18**, handles the interactions between the control object, *JoinProjectControl*, the corresponding boundary objects, *EnrollmentOption* and *JoinCompleteNotice*, and the entity object, *Project*. The control object here manages **UC-12**, JoinProject.

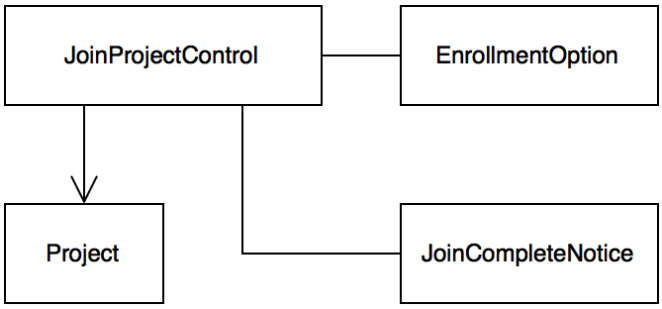


Figure 18: Control/Boundary Objects for JoinProject

Control and boundary objects for LeaveProject (UC-13):

The following figure, **Figure 19**, handles the interactions between the control object, *LeaveProjectControl*, the corresponding boundary objects, *ProjectDetailsForm* and *LeaveCompleteNotice*, and the entity object, *Project*. The control object here manages **UC-13**, LeaveProject.

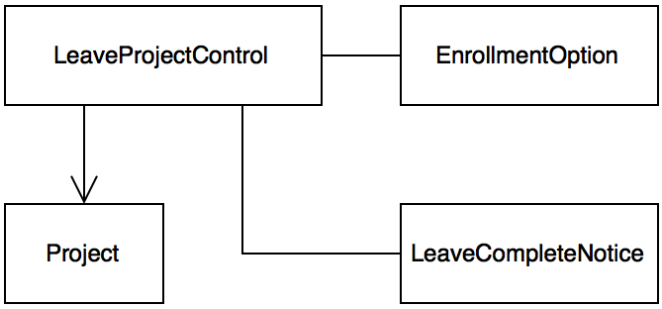


Figure 19: Control/Boundary Objects for LeaveProject

Control and boundary objects for EditPersonalValues (UC-14):

The following figure, **Figure 20**, handles the interactions between the control object, *EditPersonalValues*, the corresponding boundary objects, *ProfileOption* and *UpdateCompleteNotice*, and the entity object, *Student*. The control object here manages **UC-14**, EditPersonalValues.

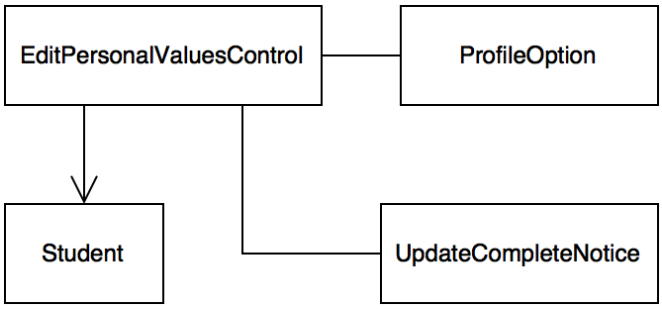


Figure 20: Control/Boundary Objects for EditPersonalValues

Control and boundary objects for EditDesiredValues (UC-15):

The following figure, **Figure 21**, handles the interactions between the control object, *EditDesiredValues*, the corresponding boundary objects, *ProfileOption* and *UpdateCompleteNotice*, and the entity object, *Student*. The control object here manages **UC-15**, EditDesiredValues.

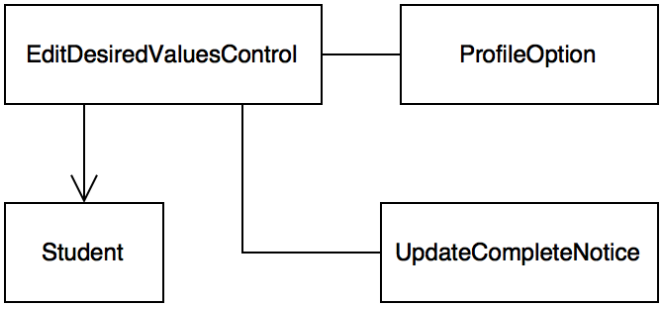


Figure 21: Control/Boundary Objects for EditDesiredValues

Control and boundary objects for InvalidInputError (UC-16):

The following figure, **Figure 22**, handles the interactions between the control object, *InvalidInputErrorControl* and the corresponding boundary objects, *InputOption* and *InvalidInputErrorNotice*. The control object here manages **UC-16**, InvalidInputError.

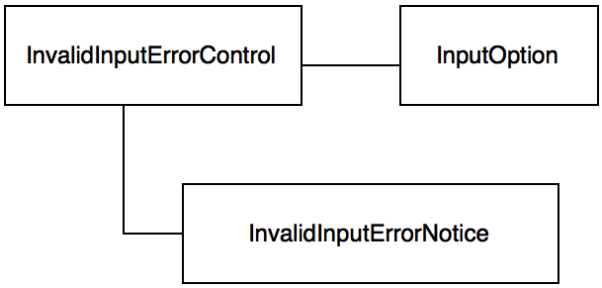


Figure 22: Control/Boundary Objects for InvalidInputError

Control and boundary objects for EditProjectName (UC-17):

The following figure, **Figure 23**, handles the interactions between the control object, *EditProjectNameControl*, the corresponding boundary objects, *EditProjectNameOption* and *ProjectNameUpdateNotice*, and the entity object, *Project*. The control object here manages **UC-17**, EditProjectName.

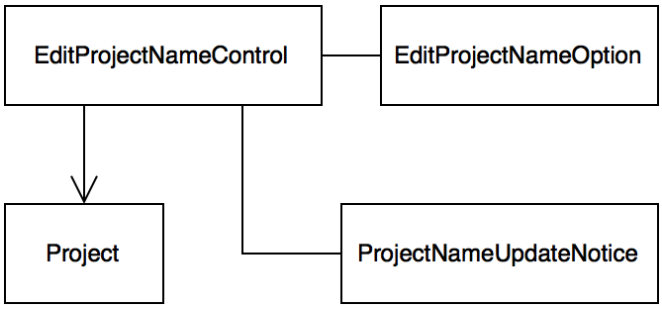


Figure 23: Control/Boundary Objects for EditProjectName

Control and boundary objects for ProjectExistsError (UC-18):

The following figure, **Figure 24**, handles the interactions between the control objects, *EditProjectDetailsControl* and *ProjectExistsErrorControl*, and the corresponding boundary object, *ProjectExistsErrorNotice*. The control object here manages **UC-18**, ProjectExistsError.

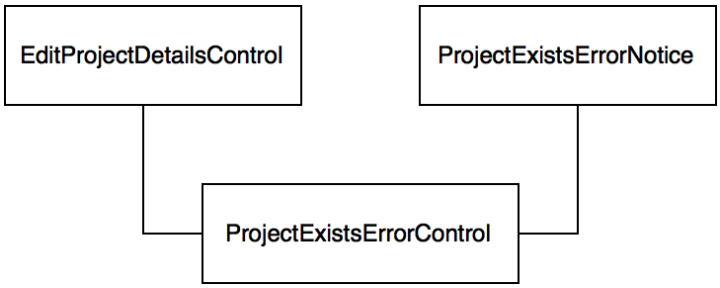


Figure 24: Control/Boundary Objects for ProjectExistsError

Control and boundary objects for EditGroupSize (UC-19):

The following figure, **Figure 25**, handles the interactions between the control object, *EditGroupSizeControl*, the corresponding boundary objects, *EditGroupSizeOption* and *GroupSizeUpdateNotice*, and the entity object, *Project*. The control object here manages **UC-19**, EditGroupSize.

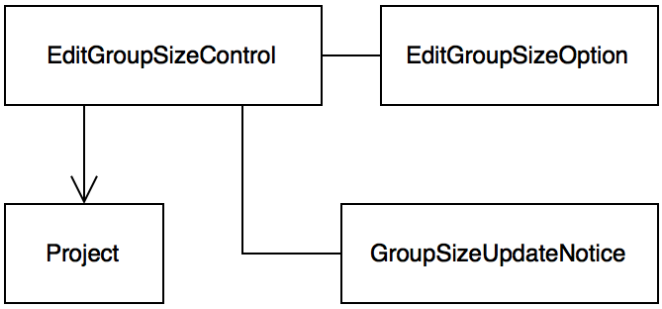


Figure 25: Control/Boundary Objects for EditGroupSize

Control and boundary objects for InvalidGroupSizeError (UC-20):

The following figure, **Figure 26**, handles the interactions between the control objects, *EditGroupSizeControl* and *InvalidGroupSizeErrorControl*, and the corresponding boundary object, *InvalidGroupSizeErrorNotice*. The control object here manages **UC-20**, InvalidGroupSizeError.

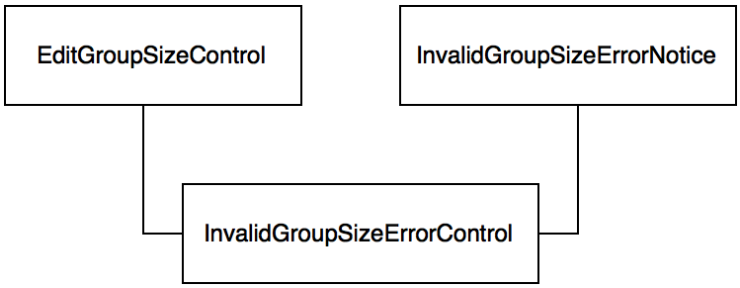


Figure 26: Control/Boundary Objects for InvalidGroupSizeError

Control and boundary objects for StorageError (UC-21):

The following figure, **Figure 27**, handles the interactions between the control objects, *StorageControl* and *StorageErrorControl*, and the corresponding boundary object, *StorageErrorNotice*. The control object here manages **UC-21**, StorageError.

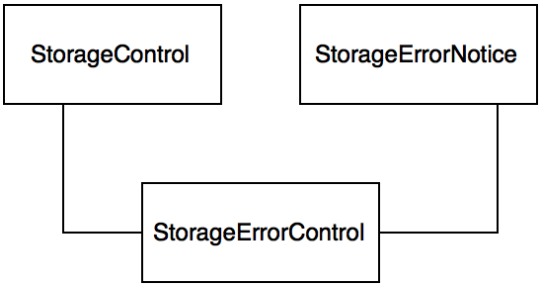


Figure 27: Control/Boundary Objects for StorageError

Control and boundary objects for StorageReadError (UC-22):

The following figure, **Figure 28**, handles the interactions between the control objects, *StorageControl* and *StorageReadErrorControl*, and the corresponding boundary object, *StorageReadErrorNotice*. The control object here manages **UC-22**, StorageReadError.

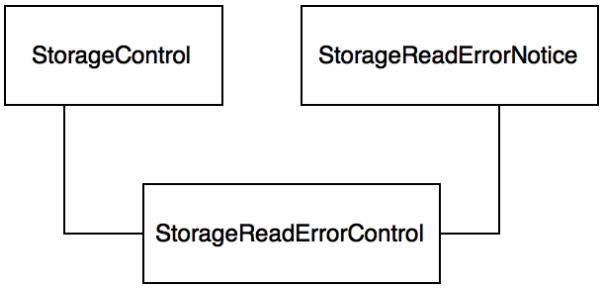


Figure 28: Control/Boundary Objects for StorageReadError

Control and boundary objects for StorageOpenError (UC-23):

The following figure, **Figure 29**, handles the interactions between the control objects, *StorageControl* and *StorageOpenErrorControl*, and the corresponding boundary object, *StorageOpenErrorNotice*. The control object here manages **UC-23**, StorageOpenError.

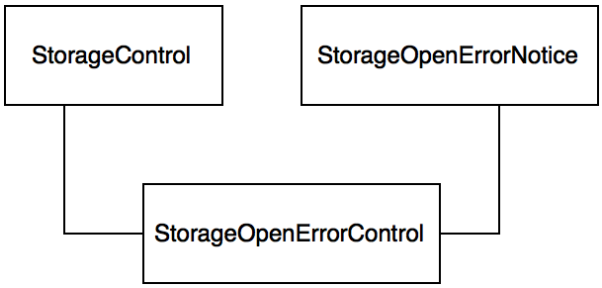


Figure 29: Control/Boundary Objects for StorageOpenError

Control and boundary objects for StorageWriteError (UC-24):

The following figure, **Figure 30**, handles the interactions between the control objects, *StorageControl* and *StorageWriteErrorControl*, and the corresponding boundary object, *StorageWriteErrorNotice*. The control object here manages **UC-24**, StorageWriteError.

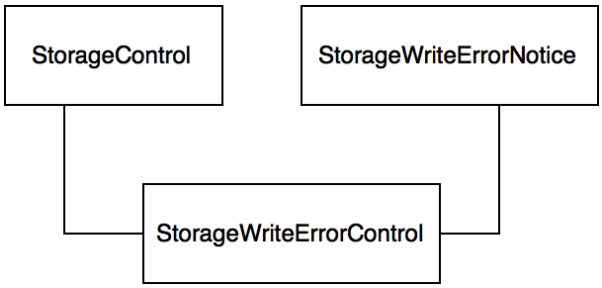


Figure 30: Control/Boundary Objects for StorageWriteError

The following table, **Table 9**, is the Data Dictionary for the control objects. It contains all the control objects shown in the above diagrams, each coupled with a very detailed definition. This definition includes details such as what use case each control object manages and when/how each control objects are created.

Table 9: Control Object Data Dictionary

ID	Control Object	Definition
CL-07	**Manage-Enrollment-Control**	The **ManageEnrollmentControl** object manages the ManageEnrollment use case. This object is created when the Student user selects the ManageEnrollmentOption. It displays a list of all the available Projects for the Student to choose from. Upon choosing, the Student will then be given the options of either joining or leaving a project. Applies to **UC-01**.
CL-08	**EditProfile-Control**	The **EditProfileControl** object manages the EditProfile use case. This object is created when the Student user selects the EditProfileOption. It provides the Student the option to edit/modify either their personal values or their desired values for their Profile. Depending on the response, control is then handed over to either the EditPersonalValuesControl or the EditDesiredValuesControl objects. Applies **UC-02**.
CL-09	**ManageProject-Control**	The **ManageProjectControl** object manages the ManageProject use case. This object is created after the Administrator user selects the ManageProjectOption. It provides the Administrator the option to select a Project and modify details for that Project. Depending on which detail is selected, control is then handed over to either the LaunchPPIDControl, the ViewPPIDResultsControl, or the EditProjectControl objects. Applies to **UC-03**.
CL-10	**CreateNew-ProjectControl**	The **CreateNewProjectControl** object manages the CreateNewProject use case. This object is created after the Administrator user selects the CreateNewProjectOption. It provides the Administrator with the ability to create new Projects, including the filling in of the details for this new project, which is managed by the EditProjectControl object. The notice, ProjectCreateNotice, is then displayed. Applies to **UC-04**.
CL-11	**LaunchPPID-Control**	The **LaunchPPIDControl** object manages the LaunchPPID use case. This object is created after the Administrator launches the PPID from the ManageProjectOption. It cycles through the list of currently enrolled students in the Project and sorts them into teams. The result is then saved to the Storage and a notice, PPIDCompleteNotice, is displayed. Applies to **UC-05**.

ID	Control Object	Definition
CL-12	**InsufficientStudentsErrorControl**	The **InsufficientStudentsErrorControl** object manages the InsufficientStudentsError use case. The object is created when the Administrator user tries to run the PPID when the amount of Students enrolled is less than the minimum team size specified for the Project. A notice, InsufficientStudentsErrorNotice, is displayed and the operation is cancelled. Applies to **UC-06**.
CL-13	**ViewPPIDResultsControl**	The **ViewPPIDResultsControl** object manages the ViewPPIDResults use case. The object is created when the Administrator user selects the ViewPPIDResultsOption. It provides the Administrator with two options for displaying the results of the last run PPID - either in simple form, where the groups are simply printed out, or in detailed form, where the reasoning for the groups is also printed out. Depending on which gets selected, control is then handed off to either the ViewPPIDSummaryControl or the ViewPPIDDetailsControl objects. Applies to **UC-07**.
CL-14	**ViewPPIDSummaryControl**	The **ViewPPIDSummaryControl** object manages the ViewPPIDSummary use case. The object is created when the Administrator user selects Summary from the two options provided in the ResultsOption object. It displays to the Administrator a print out of all the groups that the last run PPID created in a simple, easy to read manner. Applies to **UC-08**.
CL-15	**ViewPPIDDetailsControl**	The **ViewPPIDDetailsControl** object manages the ViewPPIDDetails use case. The object is created when the Administrator user selects Detailed from the two options provided in the ResultsOption object. It displays to the Administrator a detailed print out of the reasoning and steps with regards to the last PPID run. Applies to **UC-09**.
CL-16	**EditProjectDetailsControl**	The **EditProjectDetailsControl** object manages the EditProjectDetails use case. The object is created after the Administrator user inputs the new details into the ProjectDetailsForm and submits it. It then updates the values in the Project object and saves the updated Project to the Storage. A notice, DetailsUpdatedNotice, is then displayed to the Administrator. Applies to **UC-11**.
CL-17	**EditProjectControl**	The **EditProjectControl** object manages the EditProject use case. The object is created after the Administrator user selects the EditProjectOption. It then creates a ProjectDetailsForm for the user to fill in. Control is then passed over to the EditProjectDetailsControl object. Applies to **UC-10**.

ID	Control Object	Definition
CL-18	**JoinProject-Control**	The **JoinProjectControl** object manages the JoinProject use case. The object is created when the Student user selects Join from the two options provided in the EnrollmentOption object. It allows the Student to join an existing Project, then updates that Project and saves the updated Project to the Storage. Finally, a notice, JoinCompleteNotice, is displayed to the Student. Applies to **UC-12**.
CL-19	**LeaveProject-Control**	The **LeaveProjectControl** object manages the LeaveProject use case. The object is created when the Student user selects Leave from the two options provided in the EnrollmentOption object. It allows the Student to leave a Project that they are already a part of, then updates that Project and saves the updated Project to the Storage. Finally, a notice, LeaveCompleteNotice, is displayed to the Student. Applies to **UC-13**.
CL-20	**EditPersonal-ValuesControl**	The **EditPersonalValuesControl** object manages the EditPersonalValues use case. The object is created when the Student selects PersonalValues from the two options provided by the ProfileOption object. It allows the Student to input new responses for the Personal Values section of their Profile, then updates the Student object and saves the updated Student object to the Storage. Finally, a notice, UpdateCompleteNotice, is displayed to the Student. Applies to **UC-14**.
CL-21	**EditDesired-ValuesControl**	The **EditDesiredValuesControl** object manages the EditDesiredValues use case. The object is created when the Student selects DesiredValues from the two options provided by the ProfileOption object. It allows the Student to input new response for the Desired Values section of their Profile, then updates the Student object and saves the updated Student object to the Storage. Finally, a notice, UpdateCompleteNotice, is displayed to the Student. Applies to **UC-15**.
CL-22	**InvalidInput-ErrorControl**	The **InvalidInputErrorControl** object manages the InvalidInputError use case. The object is created when the Administrator inputs invalid data. It creates a notice, InvalidInputErrorNotice, and displays it to the Administrator. The input operation is then cancelled. Applies to **UC-16**.
CL-23	**EditProjectName-Control**	The **EditProjectNameControl** object manages the EditProjectName use case. The object is created when the Administrator selects the EditProjectNameOption. It then updates the Project object and saves the updated Project object to the Storage. Finally, a notice, ProjectNameUpdateNotice, is displayed to the Administrator. Applies to **UC-17**.

ID	Control Object	Definition
CL-24	**ProjectExists-ErrorControl**	The **ProjectExistsErrorControl** object manages the ProjectExistsError use case. The object is created when Administrator users attempt to set the Project name to a name that alread exists. A notice, ProjectExistsErrorNotice, is displayed to the Administrator, and the EditProjectName operation is cancelled. Applies to **UC-18**.
CL-25	**EditGroupSize-Control**	The **EditGroupSizeControl** object manages the EditGroupSize use case. The object is created when the Administrator user selects the EditGroupSizeOption. It allows the Administrator to input a new Maximum and Minimum group size value and updates the Project object. Finally, a notice, GroupSizeUpdateNotice, is displayed to the Administrator. Applies to **UC-19**.
CL-26	**InvalidGroupSize-ErrorControl**	The **InvalidGroupSizeErrorControl** object manages the InvalidGroupSizeError use case. The object is created when the Administrator attempts to input an invalid size for the Maximum and Minimum group size variables. A notice, InvalidGroupSizeErrorNotice, is displayed to the Administrator, and the EditGroupSize operation is cancelled. Applies to **UC-20**.
CL-27	**StorageControl**	The **StorageControl** object manages interaction with the storage, and is responsible for delegating to the **StorageErrorControl**. Applies to **UC-21**, **UC-22**, **UC-23**, **UC-24**.
CL-28	**StorageError-Control**	The **StorageErrorControl** object manages the StorageError use case. The object is created when any user makes a request to the Storage and the request cannot be met. It then displays a notice, StorageErrorNotice, to the user. Applies to **UC-21**.
CL-29	**StorageRead-ErrorControl**	The **StorageReadErrorControl** object manages the StorageReadError use case. The object is created when any user makes a request to read a file from the Storage and the read operation fails. It then displays a notice, StorageReadErrorNotice, to the user. Applies to **UC-22**.
CL-30	**StorageOpen-ErrorControl**	The **StorageOpenErrorControl** object manages the StorageOpenError use case. The object is created when any user makes a request to open the Storage and the operation fails. It then displays a notice, StorageOpenErrorNotice, to the user. Applies to **UC-23**.
CL-31	**StorageWrite-ErrorControl**	object manage the StorageWriteError use case. The object is created when any user makes a write to the Storage request and the operation fails. It then displays a notice, StorageWriteErrorNotice, to the user. Applies to **UC-24**.

The following table, **Table 10**, is the Data Dictionary for the boundary objects. It contains all the boundary objects shown in the above diagrams, each coupled with a detailed definition. In essence, the definition explains what each object does.

Table 10: Boundary Object Data Dictionary

ID	Boundary Object	Definition
CL-32	**Manage-EnrollmentOption**	This object allows the Student to initiate the management of their enrollment in Projects. Applies to **UC-01**.
CL-33	**ProjectListNotice**	This object is responsible for the visual output of the list of Projects to the User. It gets the required information from the Project objects and displays it. Applies to **UC-01**, **UC-03**.
CL-34	**EnrollmentOption**	This object allows the Student to view and choose between the two enrollment options: Join or Leave. Applies to **UC-01**, **UC-12**, **UC-13**.
CL-35	**ProjectSelection-Option**	This object reads the selected Project from the Administrator. Applies to **UC-01**, **UC-03**.
CL-36	**EditProfileOption**	This object allows the Student to initiate the editing of their profile. Applies to **UC-02**.
CL-37	**ProfileOption**	This object allows the Student to view and choose between the two profile options: edit Personal Details or edit Desired Details. Applies to **UC-02**, **UC-14**, **UC-15**.
CL-38	**ManageProject-Option**	This object allows the Administrator to initiate the management of a Project. Applies to **UC-03**.
CL-39	**ManageOption**	This object allows the Administrator to view and choose between two managing options: view the results of the last PPID, or edit a project. Applies to **UC-03**.
CL-40	**CreateNew-ProjectOption**	This object allows the Administrator to initiate the process of creating a new Project. Applies to **UC-04**.
CL-41	**ProjectCreate-Notice**	This object displays a notice to the Administrator stating that the Project has been successfully created. Applies to **UC-04**.
CL-42	**ProjectDetails-Form**	This object is a form that reads the inputted values from the Administrator for the Project details. Applies to **UC-04**, **UC-10**, **UC-11**.
CL-43	**PPIDComplete-Notice**	This object displays a notice to the Administrator stating that the PPID has run successfully. Applies to **UC-05**.
CL-44	**Insufficient-StudentsError-Notice**	This object displays a notice to the Administrator stating that the PPID has failed due to an insufficient amount of students enrolled in the Project. Applies to **UC-06**.

ID	Boundary Object	Definition
CL-45	**ResultsOption**	This object allows the Administrator to view and choose between the two result options: Summary or Detailed. Applies to **UC-07**.
CL-46	**SummaryResults-Notice**	This object displays a notice to the Administrator that shows the simple summary print out of the Groups. Applies to **UC-08**.
CL-47	**DetailedResults-Notice**	This object displays a notice to the Administrator that shows the detailed print out of the Groups and the reasoning that the algorithm had when making the decisions. Applies to **UC-09**.
CL-48	**DetailsUpdated-Notice**	This object displays a notice to the Administrator stating that the saving of the new Project details to Storage was successful. Applies to **UC-11**.
CL-49	**ProjectUpdate-Notice**	This object displays a notice to the Administrator stating that the editing of the Project has been completed successfully. Applies to **UC-10**.
CL-50	**JoinComplete-Notice**	This object displays a notice to the Student stating that the joining of a Project was successful. Applies to **UC-12**.
CL-51	**LeaveComplete-Notice**	This object displays a notice to the Student stating that the leaving of a Project was successful. Applies to **UC-13**.
CL-52	**UpdateComplete-Notice**	This object displays a notice to the Student stating that the saving of the new Profile details (either Personal values or Desired values) to the Storage was successful. Applies to **UC-14**, **UC-15**.
CL-53	**InputOption**	This object reads the input from the User. Applies to **UC-16**.
CL-54	**InvalidInput-ErrorNotice**	This object displays a notice to the User stating that the input given was invalid. Applies to **UC-16**.
CL-55	**EditProjectName-Option**	This object reads the input from the Administrator for the new Project Name field. Applies to **UC-17**.
CL-56	**ProjectName-UpdateNotice**	This object displays a notice to the Administrator stating that the updating/setting of the Project name has been successful. Applies to **UC-17**.
CL-57	**ProjectExists-ErrorNotice**	This object displays a notice to the Administrator stating that the desired Project name has already been taken. Applies to **UC-18**.
CL-58	**EditGroupSize-Option**	This object reads the input from the Administrator for the Maximum and Minimum group size fields. Applies to **UC-19**.

ID	Boundary Object	Definition
CL-59	**GroupSize-UpdateNotice**	This object displays a notice to the Administrator stating that the group size has been successfully updated. Applies to **UC-19**.
CL-60	**InvalidGroupSize-ErrorNotice**	This object displays a notice to the Administrator stating that the given value for the group size is invalid. Applies to **UC-20**.
CL-61	**StorageError-Notice**	This object displays a notice to the User stating that there was an error with the request to the Storage. Applies to **UC-21**.
CL-62	**StorageRead-ErrorNotice**	This object displays a notice to the User stating that there was an error with regards to reading a file from Storage. Applies to **UC-22**.
CL-63	**StorageOpen-ErrorNotice**	This object displays a notice to the User stating that there was an error with regards to opening the Storage. Applies to **UC-23**.
CL-64	**StorageWrite-ErrorNotice**	This object displays a notice to the User stating that there was an error with regards to writing to the Storage. Applies to **UC-24**.

2.4.3 Dynamic Model

Dynamic models serve to represent how a system works or behaves from an outside point-of-view. There will be two different types of dynamic models in this section: state machines and sequence diagrams. State machines are very particular in the sense that they will only represent the behavior of a single object. For this project, only the entity state machines are shown. Sequence diagrams are different in the sense that they provide a visual representation of the interactions between multiple objects for the fulfillment of a single use case.

State Machines

State machines are a special kind of dynamic model that shows all the possible states of a single object. Also shown in here is the interaction between these different states - that is, how it goes from one state to the other. The starting and ending points, or beginning and exit points, are also shown. Specifically, these state machines only concern entity objects for this particular project.

State Machines for User/Student/Administrator:

The following diagram, **Figure 31**, represents a state machine for any type of User object. Shown here is the different states involved in logging in/logging out.

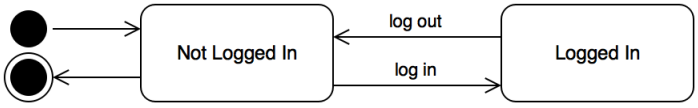


Figure 31: State Machine for User/Student/Administrator Login (SM-01)

State Machines for Student:

The following diagram, **Figure 32**, represents a state machine for the Student object. Shown here is the different states involved in the joining of Projects.

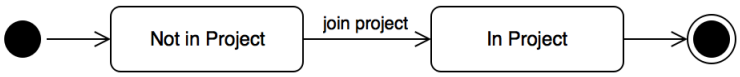


Figure 32: State Machine for Student Join Project (SM-02)

The following diagram, **Figure 33**, represents a state machine for the Student object. Shown here is the different states involved in the leaving of Projects.

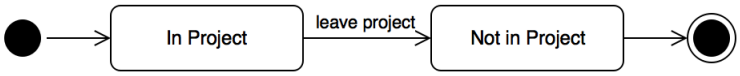


Figure 33: State Machine for Student Leave Project (SM-03)

State Machines for Profile:

The following diagram, **Figure 34**, represents a state machine for the Profile object. Shown here is the different states involved in managing profile details.

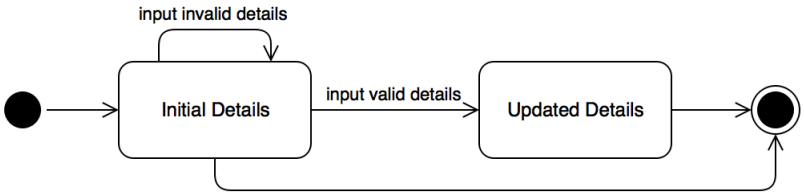


Figure 34: State Machine for Profile Details (SM-04)

State Machines for Project:

The following diagram, **Figure 35**, represents a state machine for the Project object. Shown here is the different states involved in the adding and removing Students.

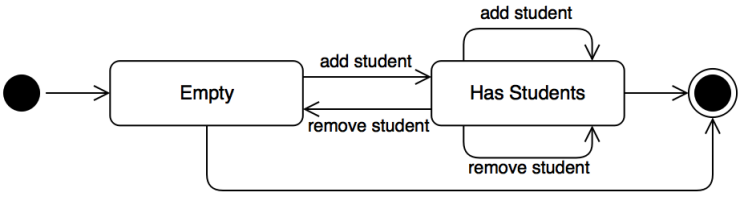


Figure 35: State Machine for Project Student Enrollment (SM-05)

The following diagram, **Figure 36**, represents a state machine for the Project object. Shown here is the different states involved in the setting of a Project's name.

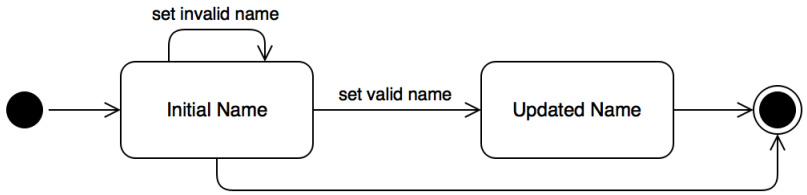


Figure 36: State Machine for Project Name (SM-06)

The following diagram, **Figure 37**, represents a state machine for the Project object. Shown here is the different states involved in showing the results of/running the PPID.

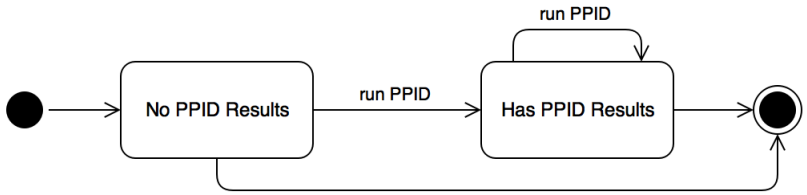


Figure 37: State Machine for Project PPID Results (SM-07)

State Machines for Group:

The following diagram, **Figure 38**, represents a state machine for the Group object. Shown here is the different states involved in adding/enrolling students into Groups.

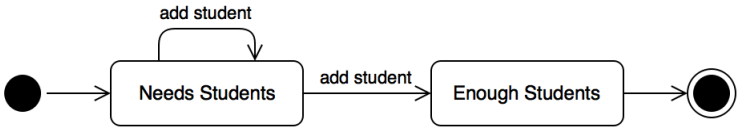


Figure 38: State Machine for Group Enrollment (SM-08)

Sequence Diagrams

Sequence diagrams are a great visual representation of the many interactions between different objects needed to satisfy one use case. Note that each object has a vertical dotted line - this represents the time-line of that object. Through the course of the diagram, multiple objects will be created and destroyed after fulfilling their purpose.

Sequence diagram for ManageEnrollment (UC-01):

The following figure, **Figure 39**, shows the sequence diagram for the *ManageEnrollment* use case. The participating actor for this use case is the Student user. This figure shows the necessary interactions between the control object, *LaunchPPIDControl*, and the entity/boundary objects in order to manage **UC-01**.

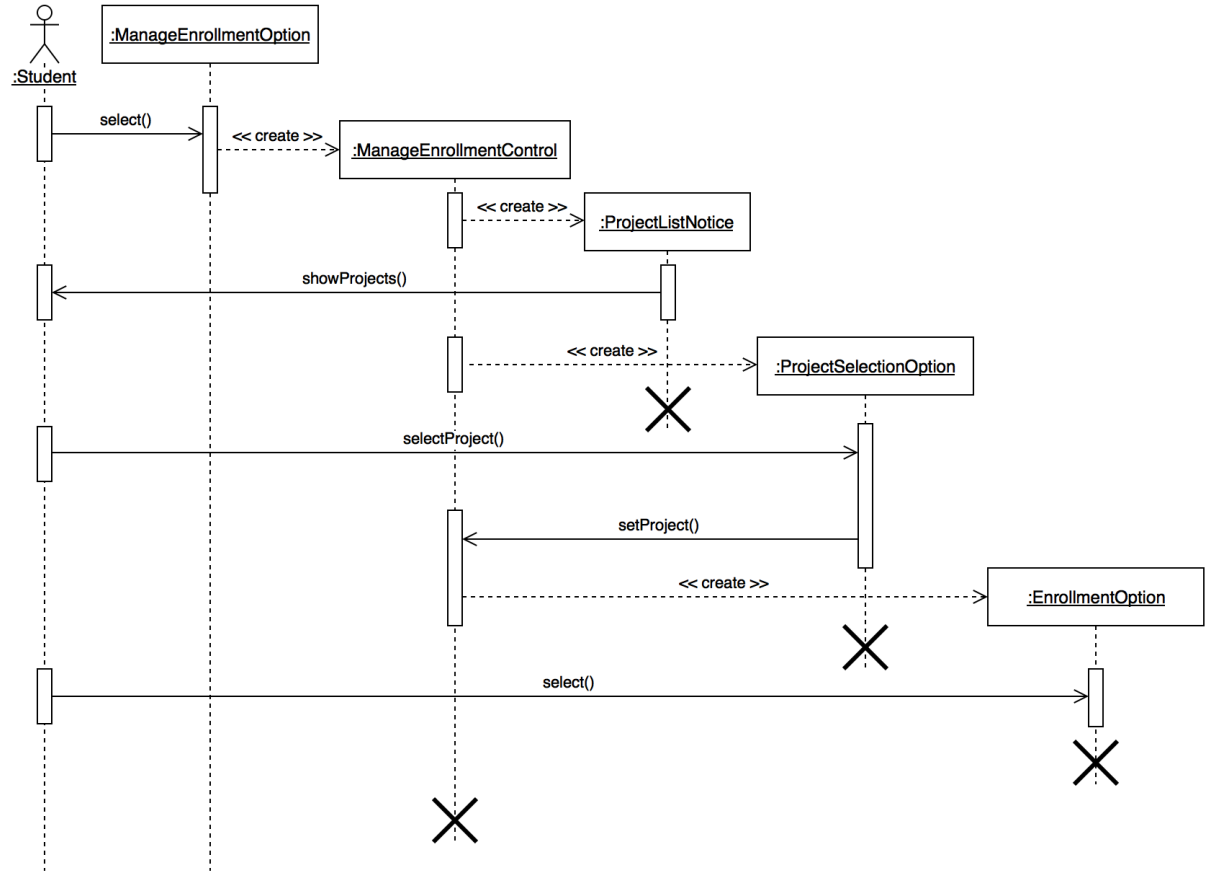


Figure 39: Sequence Diagram for ManageEnrollment (SQ-01)

Sequence diagram for EditProfile (UC-02):

The following figure, seq-edit-profile, shows the sequence diagram for the *EditProfile* use case. The participating actor is the Student user. This figure shows the necessary interactions between the control object, *EditProfileControl*, and the entity/boundary objects in order to manage **UC-02**.

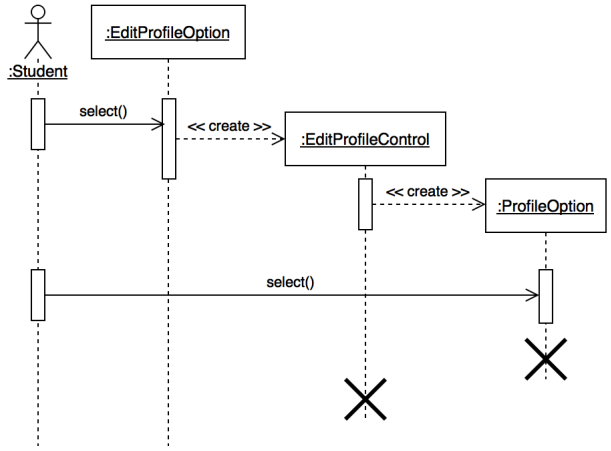


Figure 40: Sequence Diagram for EditProfile (SQ-02)

Sequence diagram for ManageProject (UC-03):

The following figure, **Figure 41**, shows the sequence diagram for the *ManageProject* use case. The participating actor is the Student user. This figure shows the necessary interactions between the control object, *ManageProjectControl*, and the entity/boundary objects in order to manage **UC-03**.

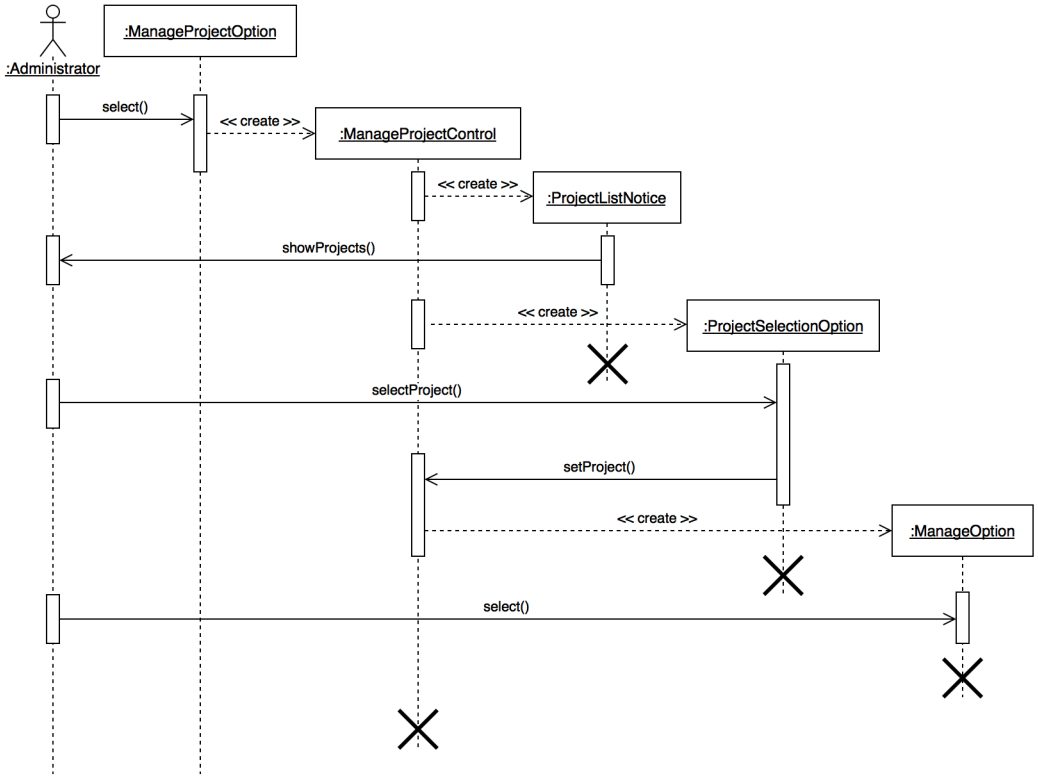


Figure 41: Sequence Diagram for ManageProject (SQ-03)

Sequence diagram for CreateNewProject (UC-04):

The following figure, **Figure 42**, shows the sequence diagram for the *CreateNewProject* use case. The participating actor for this use case is the Administrator user. This figure shows the necessary interactions between the control object, *CreateNewProjectControl*, and the entity/boundary objects in order to manage **UC-04**.

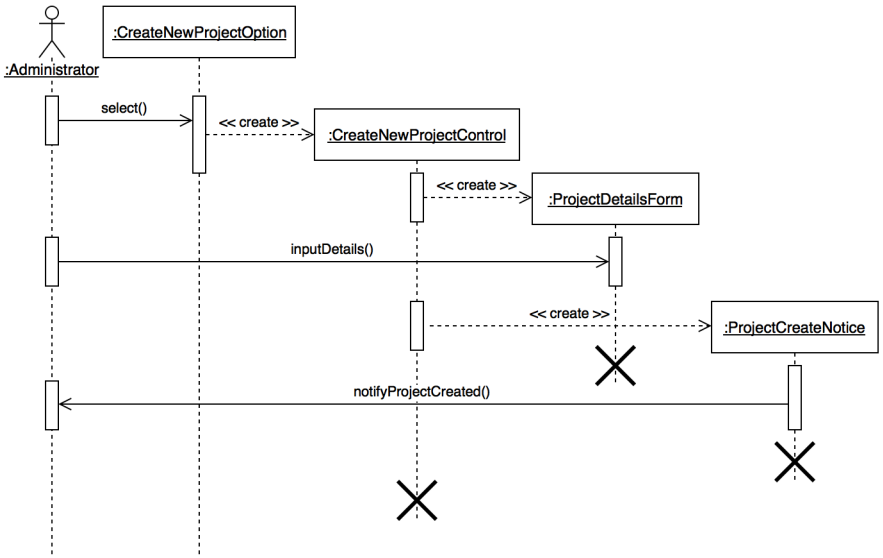


Figure 42: Sequence Diagram for CreateNewProject (SQ-04)

Sequence diagram for LaunchPPID (UC-05):

The following figure, **Figure 43**, shows the sequence diagram for the *LaunchPPID* use case. The participating actor for this use case is the Administrator user. This figure shows the necessary interactions between the control object, *LaunchPPIDControl*, and the entity/boundary objects in order to manage **UC-05**.

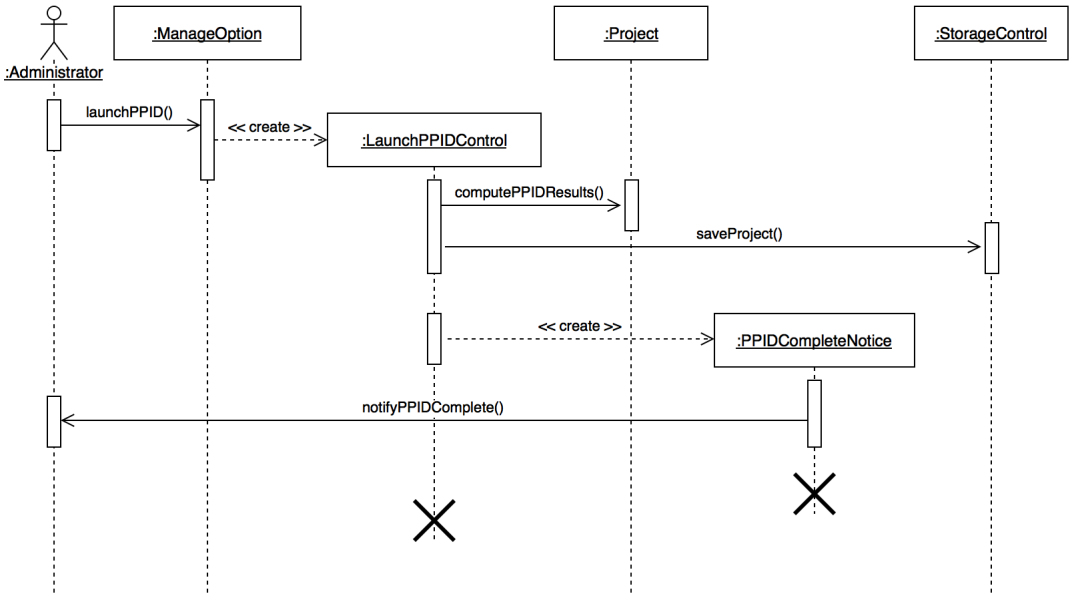


Figure 43: Sequence Diagram for LaunchPPID (SQ-05)

Sequence diagram for InsufficientStudentsError (UC-06):

The following figure, **Figure 44**, shows the sequence diagram for the *InsufficientStudentsError* use case. The participating actor for this use case is the Administrator user. This figure shows the necessary interactions between the control object, *InsufficientStudentsErrorControl*, and the entity/boundary objects in order to manage **UC-06**.

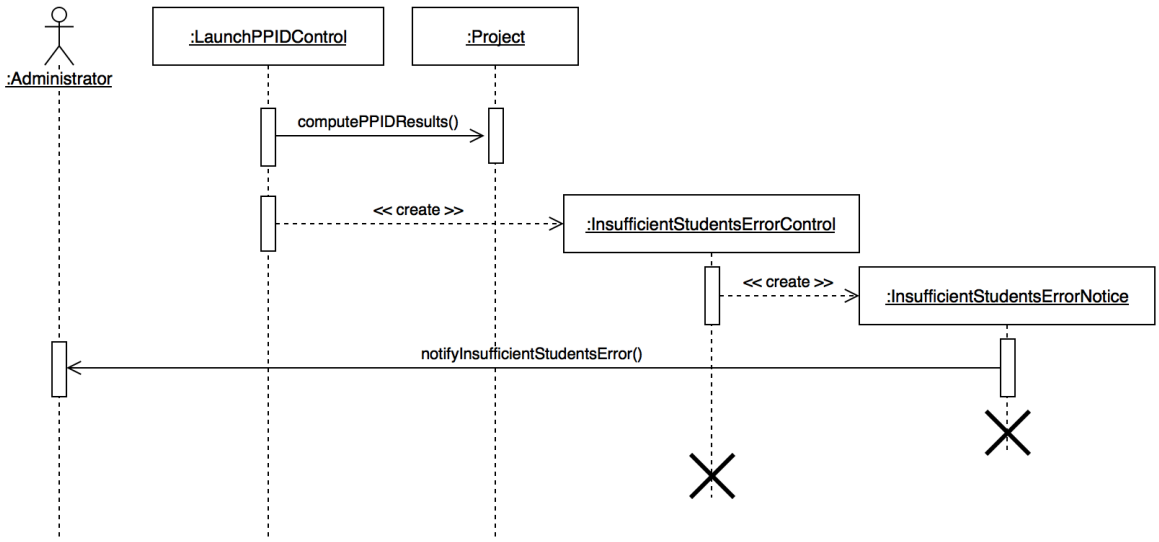


Figure 44: Sequence Diagram for InsufficientStudentsError (SQ-06)

Sequence diagram for ViewPPIDResults (UC-07):

The following figure, **Figure 45**, shows the sequence diagram for the *ViewPPIDResults* use case. The participating actor for this use case is the Administrator user. This figure shows the necessary interactions between the control object, *ViewPPIDResultsControl*, and the entity/boundary objects in order to manage **UC-07**.

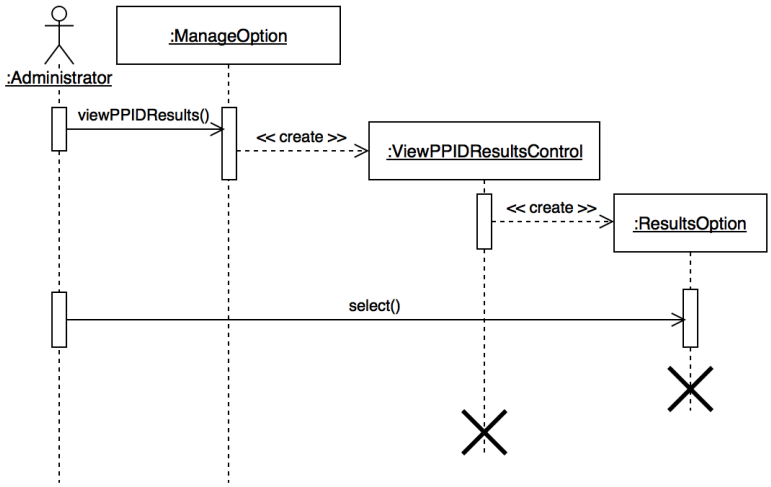


Figure 45: Sequence Diagram for ViewPPIDResults (SQ-07)

Sequence diagram for ViewPPIDSummary (UC-08):

The following figure, **Figure 46**, shows the sequence diagram for the *ViewPPIDSummary* use case. The participating actor for this use case is the Administrator user. This figure shows the necessary interactions between the control object, *ViewPPIDSummaryControl*, and the entity/boundary objects in order to manage **UC-08**.

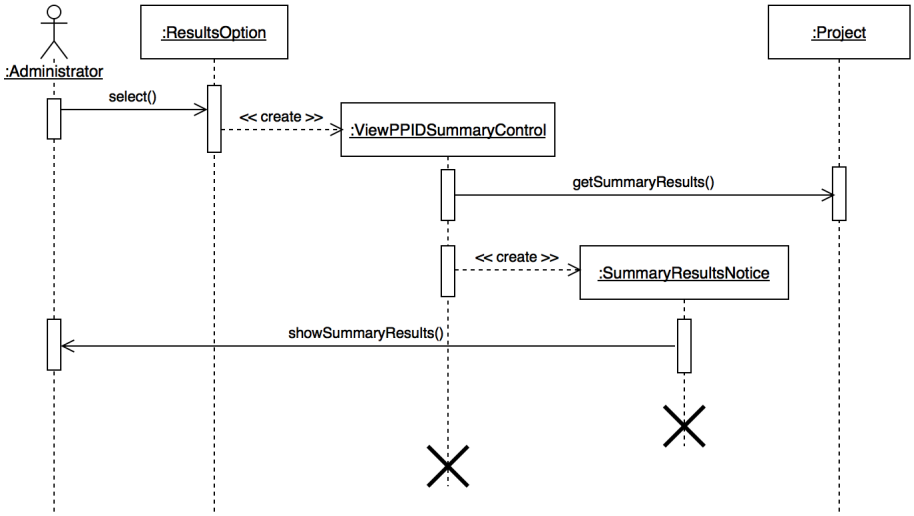


Figure 46: Sequence Diagram for ViewPPIDSummary (SQ-08)

Sequence diagram for ViewPPIDDDetails (UC-09):

The following figure, **Figure 47**, shows the sequence diagram for the *ViewPPIDDetails* use case. The participating actor for this use case is the Administrator user. This figure shows the necessary interactions between the control object, *ViewPPIDDetailsControl*, and the entity/boundary objects in order to manage **UC-09**.

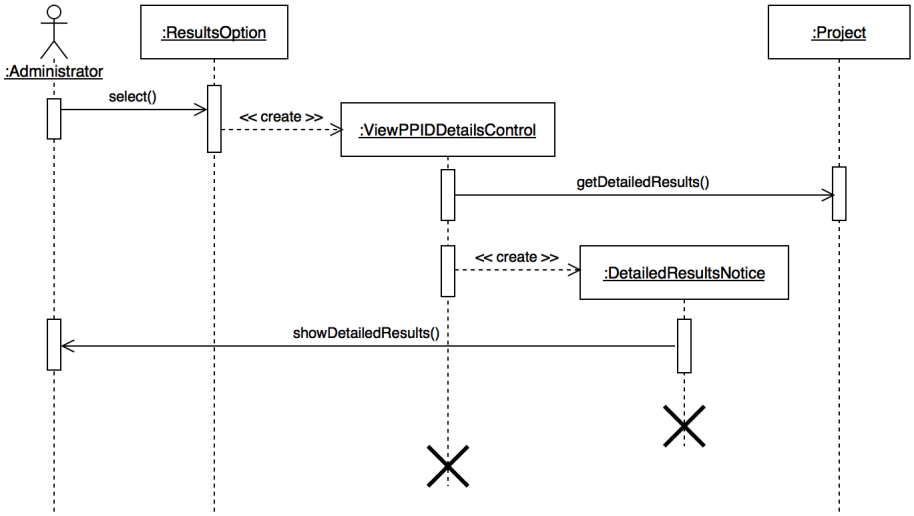


Figure 47: Sequence Diagram for ViewPPIDDetails (SQ-09)

Sequence diagram for EditProject (UC-10):

The following figure, **Figure 48**, shows the sequence diagram for the *EditProject* use case. The participating actor for this use case is the Administrator user. This figure shows the necessary interactions between the control object, *EditProjectControl*, and the entity/boundary objects in order to manage **UC-10**.

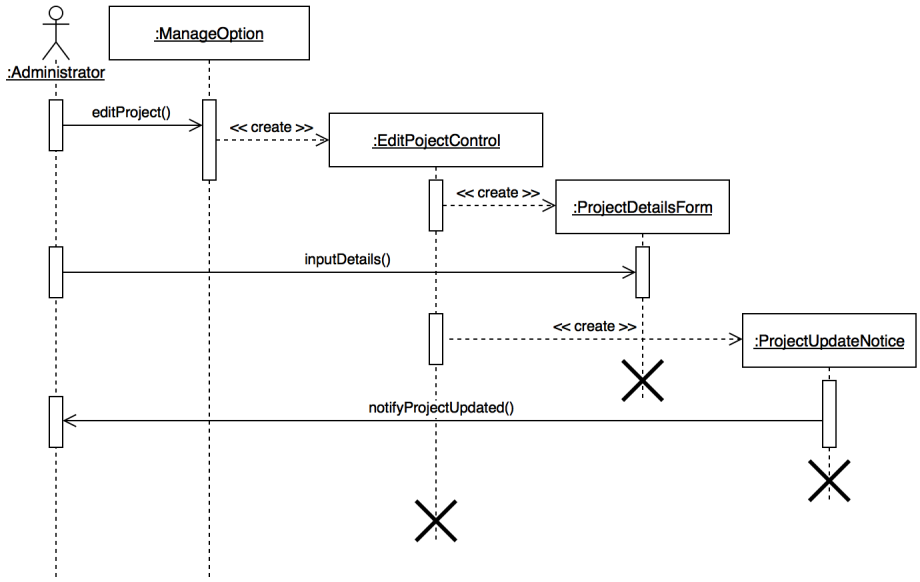


Figure 48: Sequence Diagram for EditProject (SQ-10)

Sequence diagram for EditProjectDetails (UC-11):

The following figure, **Figure 49**, shows the sequence diagram for the *EditProjectDetails* use case. The participating actor for this use case is the Administrator user. This figure shows the necessary interactions between the control object, *EditProjectDetailsControl*, and the entity/boundary objects in order to manage **UC-11**.

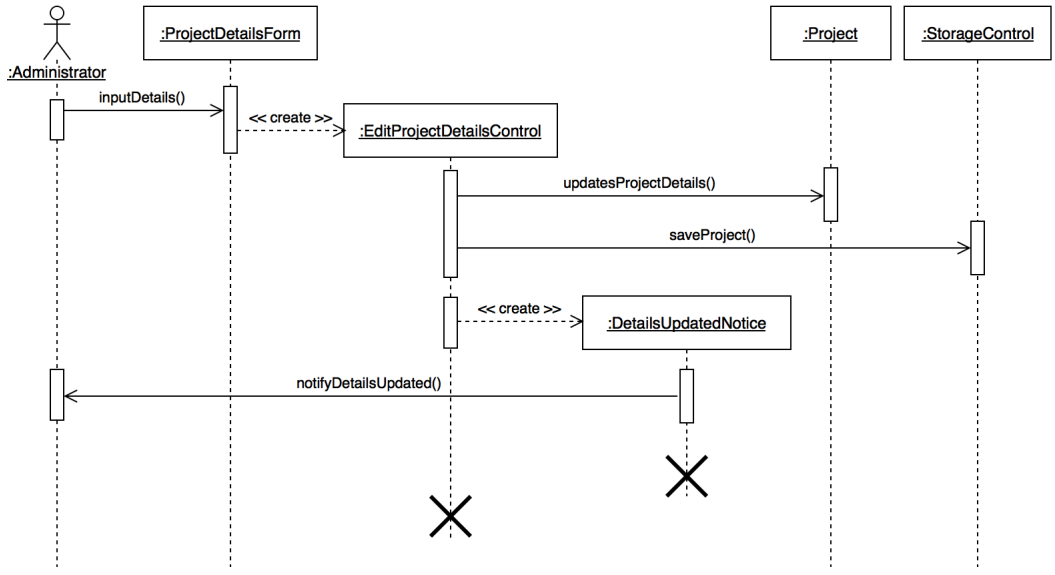


Figure 49: Sequence Diagram for EditProjectDetails (SQ-11)

Sequence diagram for JoinProject (UC-12)

The following figure, **Figure 50**, shows the sequence diagram for the *JoinProject* use case. The participating actor for this use case is the Student user. This figure shows the necessary interactions between the control object, *JoinProjectControl*, and the entity/boundary objects in order to manage **UC-12**.

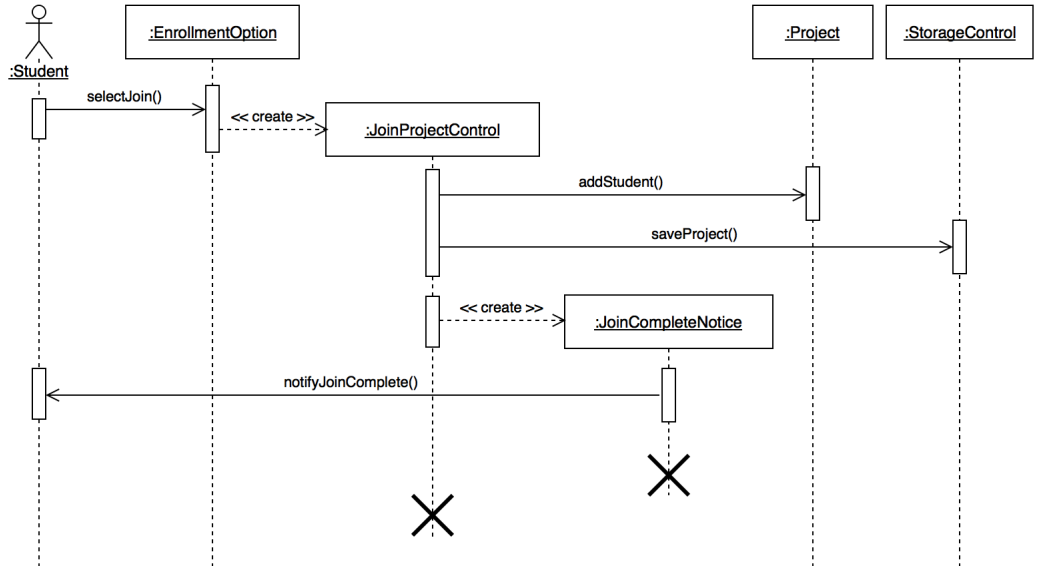


Figure 50: Sequence Diagram for JoinProject (SQ-12)

Sequence diagram for LeaveProject (UC-13)

The following figure, **Figure 51**, shows the sequence diagram for the *LeaveProject* use case. The participating actor for this use case is the Student user. This figure shows the necessary interactions between the control object, *LeaveProjectControl*, and the entity/boundary objects in order to manage **UC-13**.

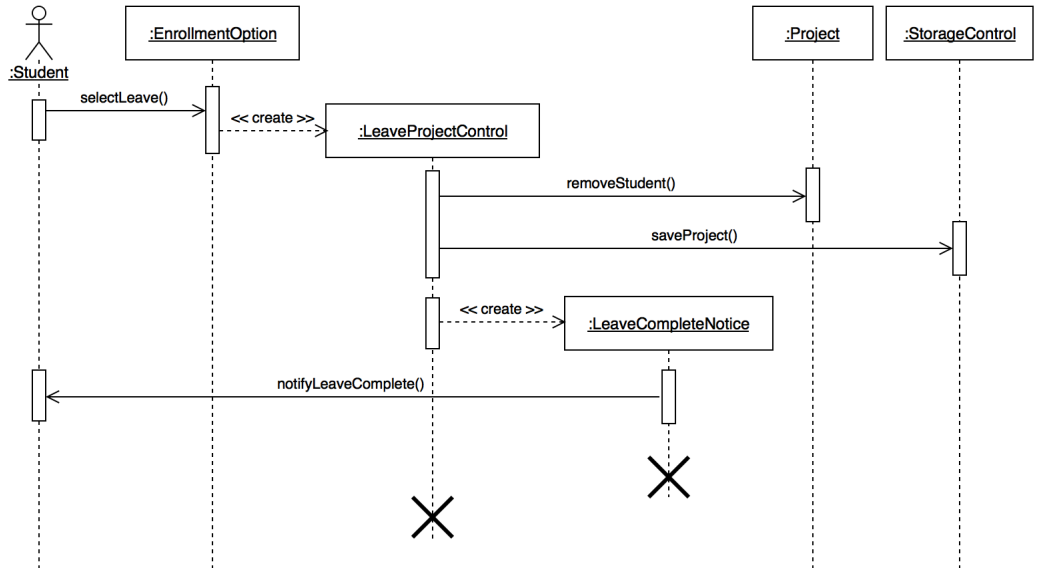


Figure 51: Sequence Diagram for LeaveProject (SQ-13)

Sequence diagram for EditPersonalValues (UC-14)

The following figure, **Figure 52**, shows the sequence diagram for the *EditPersonalValues* use case. The participating actor for this use case is the Student user. This figure shows the necessary interactions between the control object, *EditPersonalValuesControl*, and the entity/boundary objects in order to manage **UC-14**.

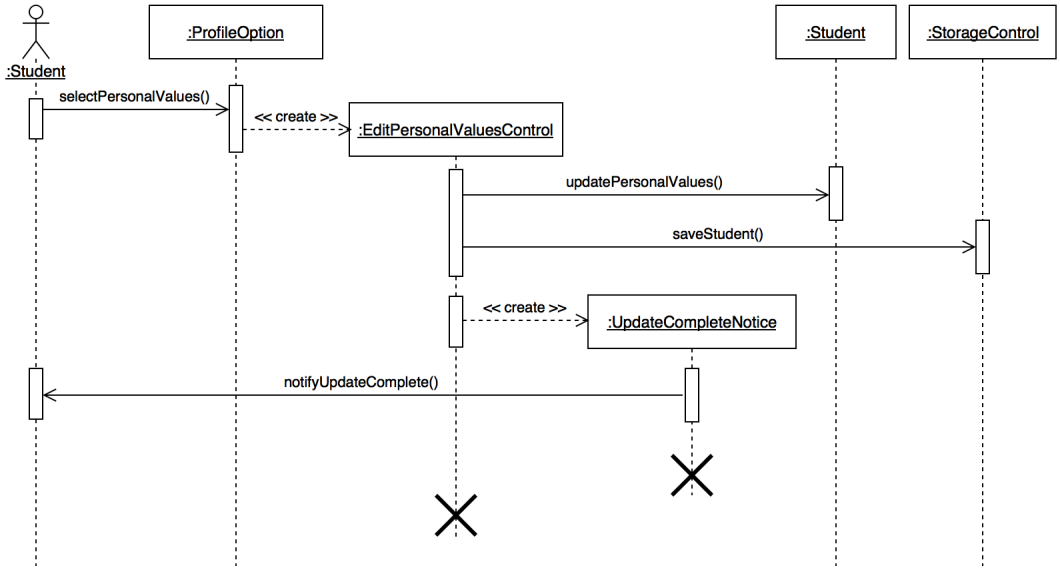


Figure 52: Sequence Diagram for EditPersonalValues (SQ-14)

Sequence diagram for EditDesiredValues (UC-15)

The following figure, **Figure 53**, shows the sequence diagram for the *EditDesiredValues* use case. The participating actor for this use case is the Student user. This figure shows the necessary interactions between the control object, *EditDesiredValuesControl*, and the entity/boundary objects in order to manage **UC-15**.

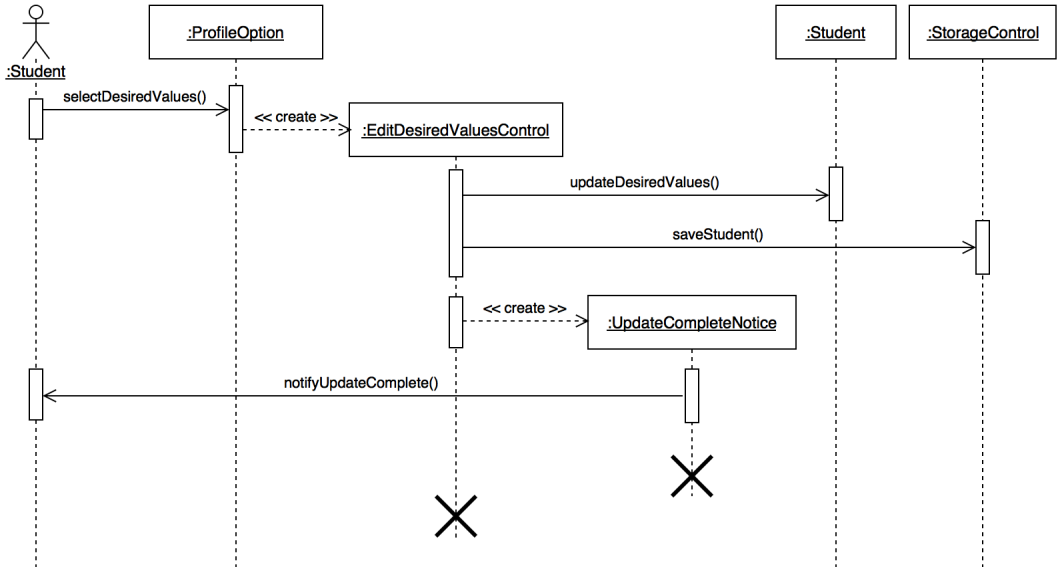


Figure 53: Sequence Diagram for EditDesiredValues (SQ-15)

Sequence diagram for InvalidInputError (UC-16)

The following figure, **Figure 54**, shows the sequence diagram for the *InvalidInputError* use case. The participating actor for this use case is the Administrator user. This figure shows the necessary interactions between the control object, *InvalidInputErrorControl*, and the entity/boundary objects in order to manage **UC-16**.

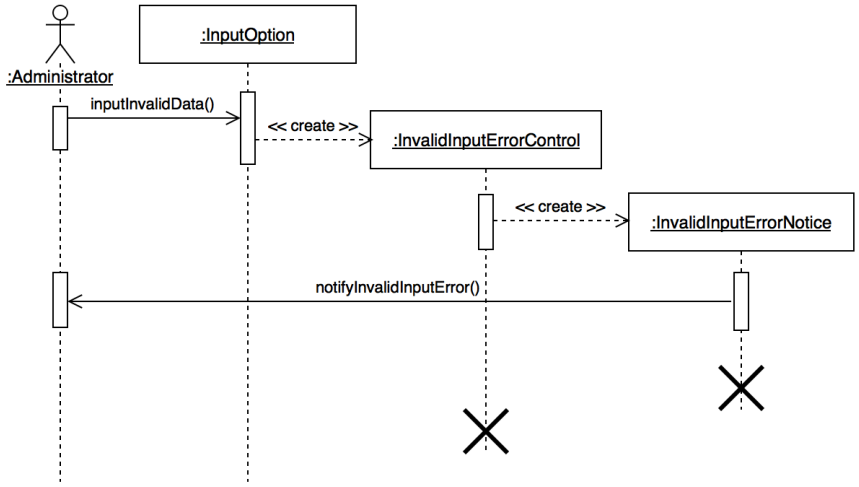


Figure 54: Sequence Diagram for InvalidInputError (SQ-16)

Sequence diagram for EditProjectName (UC-17)

The following figure, **Figure 55**, shows the sequence diagram for the *EditProjectName* use case. The participating actor for this use case is the Administrator user. This figure shows the necessary interactions between the control object, *EditProjectNameControl*, and the entity/boundary objects in order to manage **UC-17**.

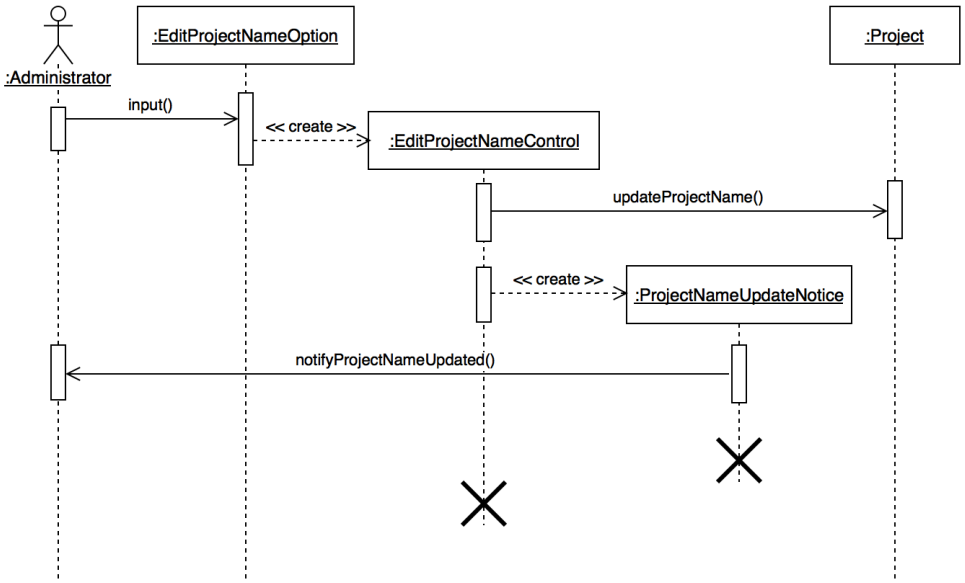


Figure 55: Sequence Diagram for EditProjectName (SQ-17)

Sequence diagram for ProjectExistsError (UC-18)

The following figure, **Figure 56**, shows the sequence diagram for the *ProjectExistsError* use case. The participating actor for this use case is the Administrator user. This figure shows the necessary interactions between the control object, *ProjectExistsErorrControl*, and the entity/boundary objects in order to manage **UC-18**.

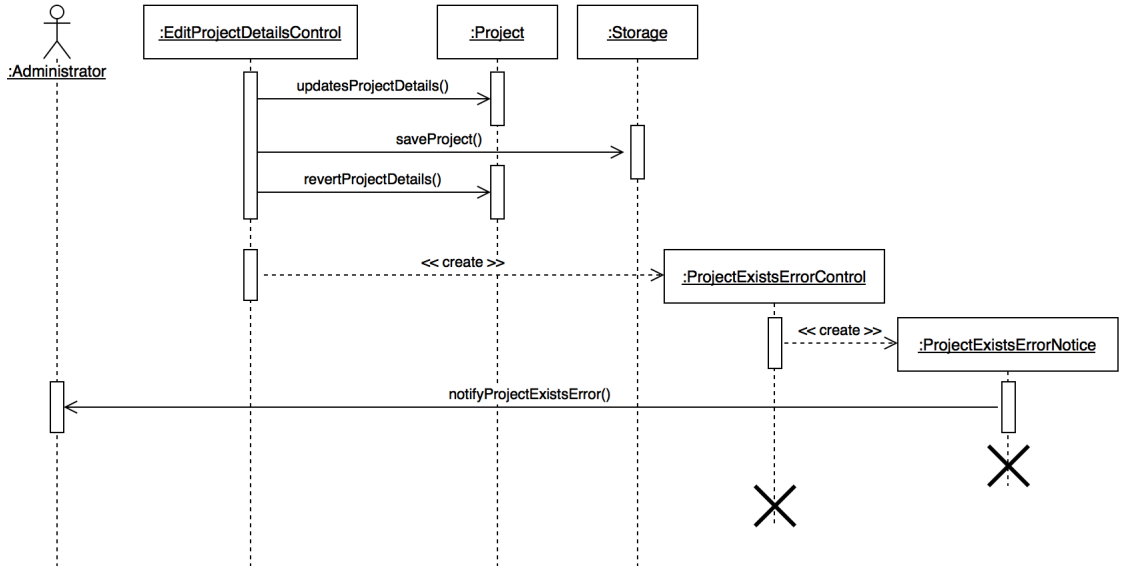


Figure 56: Sequence Diagram for ProjectExistsError (SQ-18)

Sequence diagram for EditGroupSize (UC-19)

The following figure, **Figure 57**, shows the sequence diagram for the *EditGroupSize* use case. The participating actor for this use case is the Administrator user. This figure shows the necessary interactions between the control object, *EditGroupSizeControl*, and the entity/boundary objects in order to manage **UC-19**.

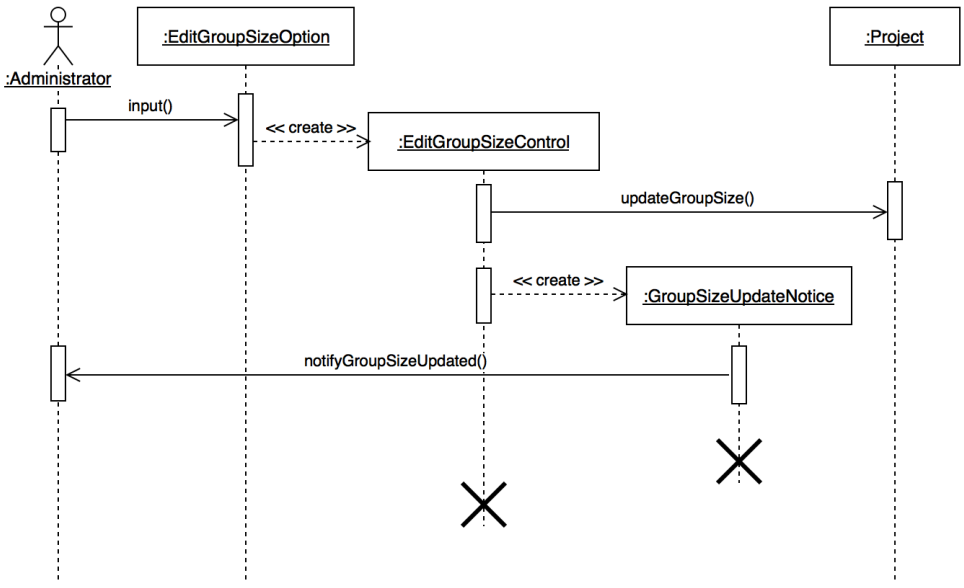


Figure 57: Sequence Diagram for EditGroupSize (SQ-19)

Sequence diagram for InvalidGroupSizeError (UC-20)

The following figure, **Figure 58**, shows the sequence diagram for the *InvalidGroupSizeError* use case. The participating actor for this use case is the Administrator user. This figure shows the necessary interactions between the control object, *InvalidGroupSizeErrorControl*, and the entity/boundary objects in order to manage **UC-20**.

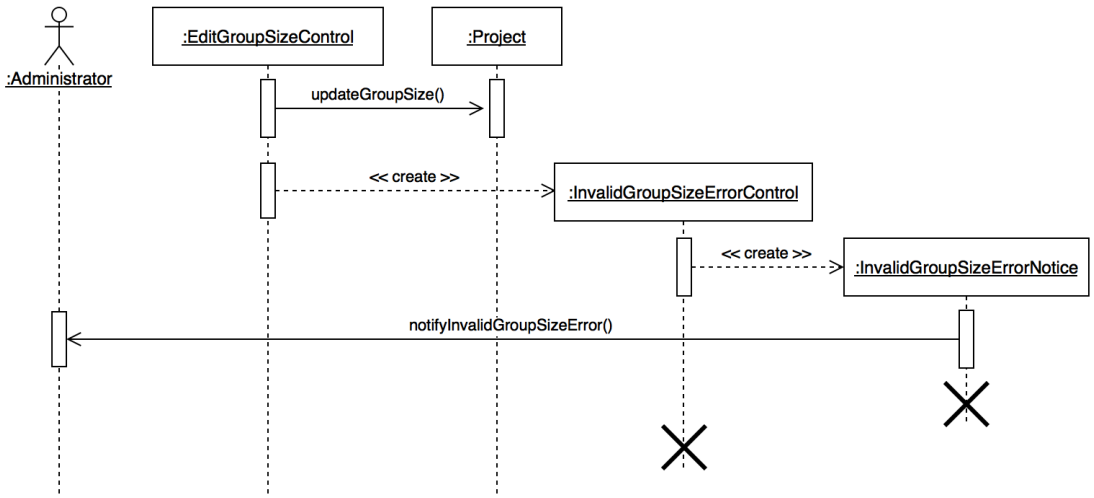


Figure 58: Sequence Diagram for InvalidGroupSizeError (SQ-20)

Sequence diagram for StorageError (UC-21)

The following figure, **Figure 59**, shows the sequence diagram for the *StorageError* use case. The participating actor for this use case is the User (both Administrator and Student). This figure shows the necessary interactions between the control object, *StorageErrorControl*, and the entity/boundary objects in order to manage **UC-21**.

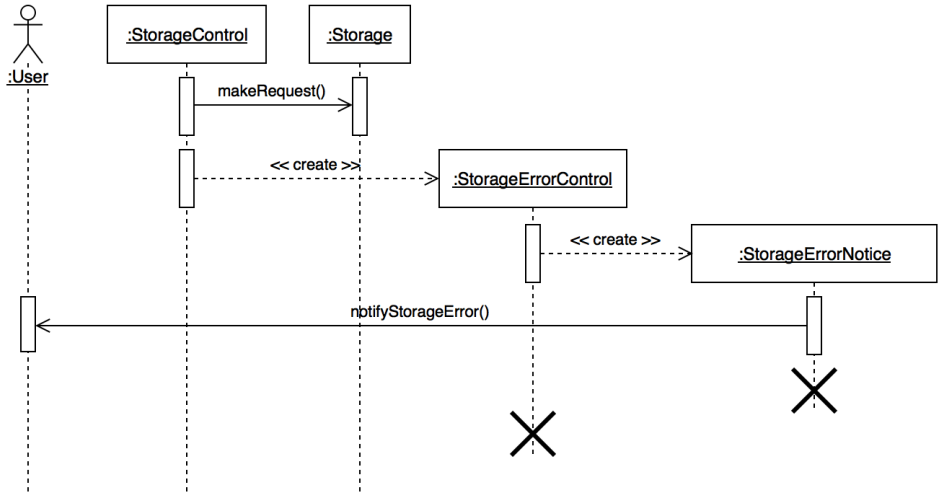


Figure 59: Sequence Diagram for StorageError (SQ-21)

Sequence diagram for StorageReadError (UC-22)

The following figure, **Figure 60**, shows the sequence diagram for the *StorageReadError* use case. The participating actor for this use case is the User (both Administrator and Student). This figure shows the necessary interactions between the control object, *StorageReadErrorControl*, and the entity/boundary objects in order to manage **UC-22**.

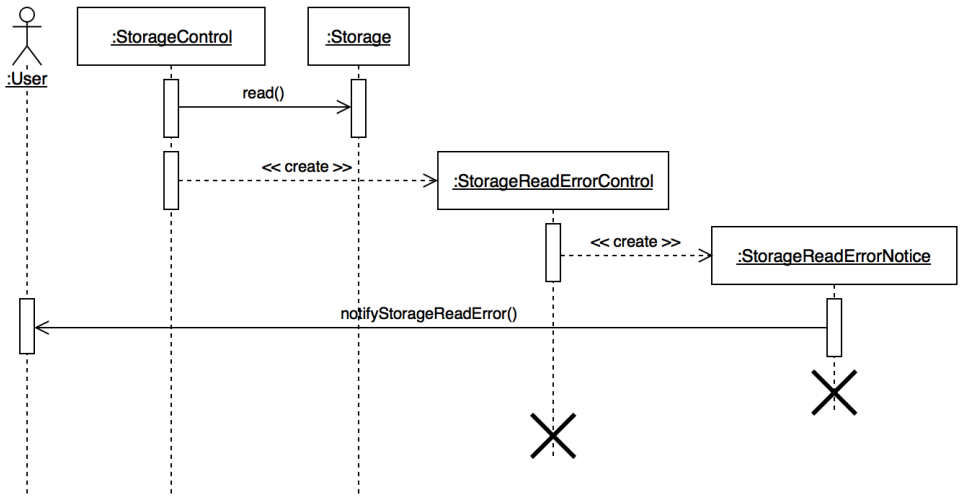


Figure 60: Sequence Diagram for StorageReadError (SQ-22)

Sequence diagram for StorageOpenError (UC-23)

The following figure, **Figure 61**, shows the sequence diagram for the *StorageOpenError* use case. The participating actor for this use case is the User (both Administrator and Student). This figure shows the necessary interactions between the control object, *StorageOpenErrorControl*, and the entity/boundary objects in order to manage **UC-23**.

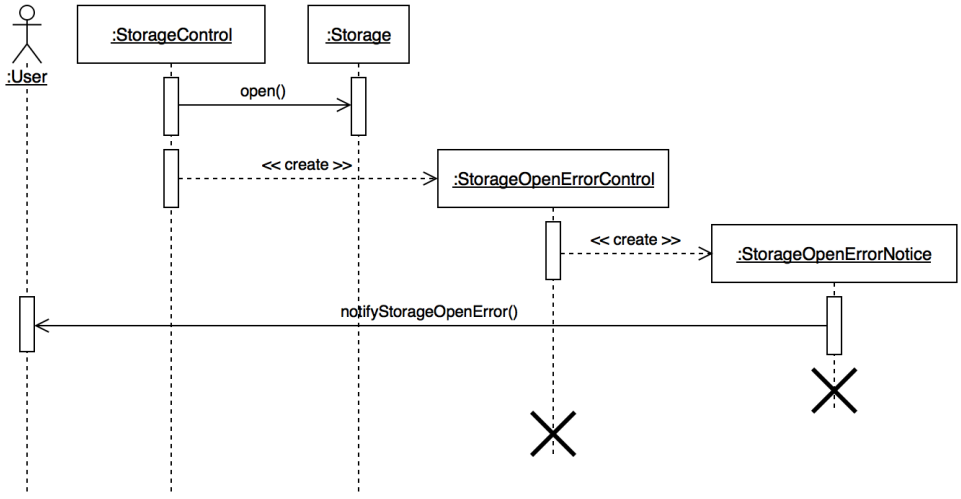


Figure 61: Sequence Diagram for StorageOpenError (SQ-23)

Sequence diagram for StorageWriteError (UC-24)

The following figure, **Figure 62**, shows the sequence diagram for the *StorageWriteError* use case. The participating actor for this use case is the User (both Administrator and Student). This figure shows the necessary interactions between the control object, *StorageWriteError*, and the entity/boundary objects in order to manage **UC-24**.

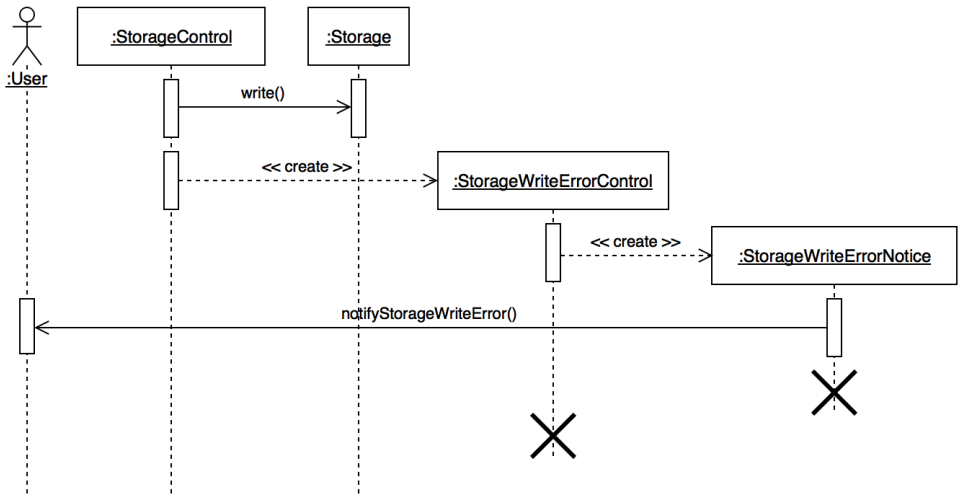


Figure 62: Sequence Diagram for StorageWriteError (SQ-24)