inVøke
Software Design Specification

Zachary Hoffman, James Kang, Sam Lundquist, Owen McEvoy, Mason Sayyadi

Table of Contents

1. SDS Revision History

Date	Author	Description
2020-01-13	SL	Imported template for the initial document.
2020-01-14	ZH	Adjusted revision history formatting and started on the software modules section and system overview sections.
2020-01-14	MS/SL	Created visual concepts for the UI of the finished product
2020-01-17	SL	Initial Draft of Software Modules finished
2020-01-17	SL	Initial Draft of Software Architecture

2020-02-02	ZH	Updates to reflect the current state of the program

2. System Overview

The Invoke system provides the ability to determine and display which students should prepare to answer and the ability to track and report this participation for an instructor to utilize in grading, feedback, and other purposes. The system is divided into the visible user interface component that the instructor uses, a cold calling data component for handling queueing the students and tracking their participation, and a data storage and reporting component that handles setup and maintenance of information between runs and the exporting of useful reports for the instructor.

3. Software Architecture

3.1 - Set of Components

The software components within this program can be broken down into three modules and eight components, they are as follows:

Instructor Interface Module Components

1. Class Roster Interface - This component is designed to display information about current classes and students to the instructor(user). It also handles user input features for the user to add/remove classes and students from the system.

2. On Deck Interface - A component dedicated to displaying information about the student queue to the user. It displays information about all of the users that are ‘on deck’ to be called on. This component also handles user input for commands that modify the state of the student queue.

Cold Calling Data Module Components

3. Class List - This includes all of the information about the different classes and the means to load in the data on the classes between sessions.

4. Class Data - This includes all of the information about a specific class and the students in that class. It also includes all of the member functions to read, write, and access this data from the Instructor Interface Module or the Data Storage/Reporting Module.

5. Student Queue Data - This component contains all of the information about the state of the student queue for a specific class including things such as who is ‘on deck.’ It also includes all of the member functions to modify the queue and register participation.

Data Storage/Reporting Module Components

6. State Persistence - This component maintains the data of the system between runs of the cold calling software. It also provides critical functions to the other Data Storage and Reporting Module components for handling the paths where both internal data and exported data is stored.

7. Roster Import/Export - A component that allows the user to import and export the class roster when necessary.

8. Reporting - Daily cold calling activity is exported into a log file and end of term reports are generated using this component. This allows the user to go back and view the previous usage for a specific day or the class in total.

3.2 - Visual Representation

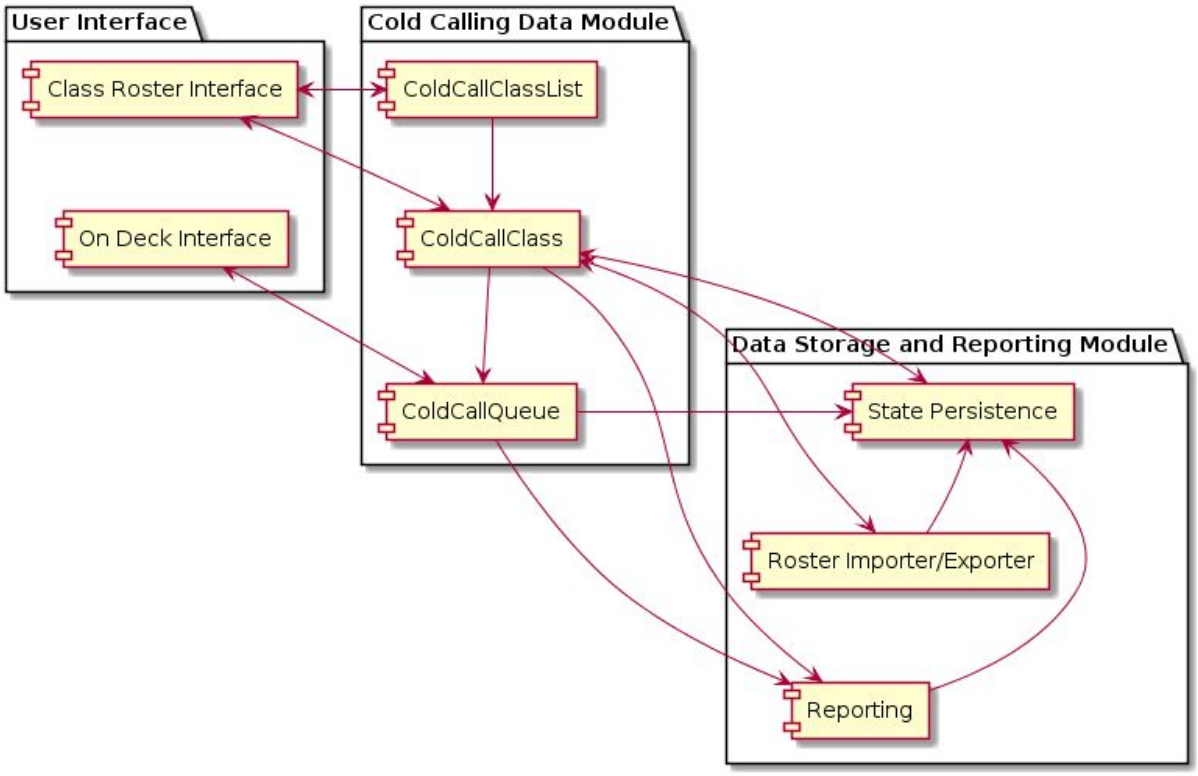


Figure 3.1: A UML component diagram to illustrate how the software components are dependent on one another.

3.3 - Architecture Design Rationale

Instructor Interface Module Components

- Class Roster Interface: We designed the UI to be organized such that the instructors can easily add and remove their classes with just a few key presses. We used a hierarchical menu approach to do this so that it organizes the desired operations neatly.
- On Deck Interface: We decided to display just a student's name in the On Deck interface to keep the interface clear of distractions.

Cold Calling Data Module Components

- Class List, Class, and Queue: The class was implemented as the basic unit of internal storage that contains individual student objects that themselves contain their participation in that class. The queue for the class is handled as a separate member of the class so that the class only needs to be concerned with maintaining the student list and not the queue positions or functions. The Class List aggregates multiple Classes worth of data so that an instructor can use the software for multiple classes.

Data Storage/Reporting Module Components

- Import/Export modules: Each function (roster updating, state maintenance, and logging and reporting) is handled by what is generally a separate component as the different functions require little to no interaction. This would allow us to add more reports and logs as needed without affecting the other components.

4. Software Modules

4.1 - Instructor Interface Module

A. The module's role and primary function.
    - The Instructor Interface Module is used to handle the IO requests of the user and also to display information about the state of the program.
    - The state of the program includes:
        - The list of classes currently in the program.
        - Information about who is on deck to be called for a specific class
    - IO requests may include:
        - Adding/removing class
        - Changing the currently displayed class
        - Removing student from on deck, optionally flagging their participation

B. The interface to other modules.
    - The Instructor Interface Module communicates directly with the Cold Calling Data Module, which has access to all of the information about the state of students and classes, as well as the state of the cold call queue for any given class. Most IO from the Instructor Interface Module is handled by the Cold Calling Data Module.

C. Models

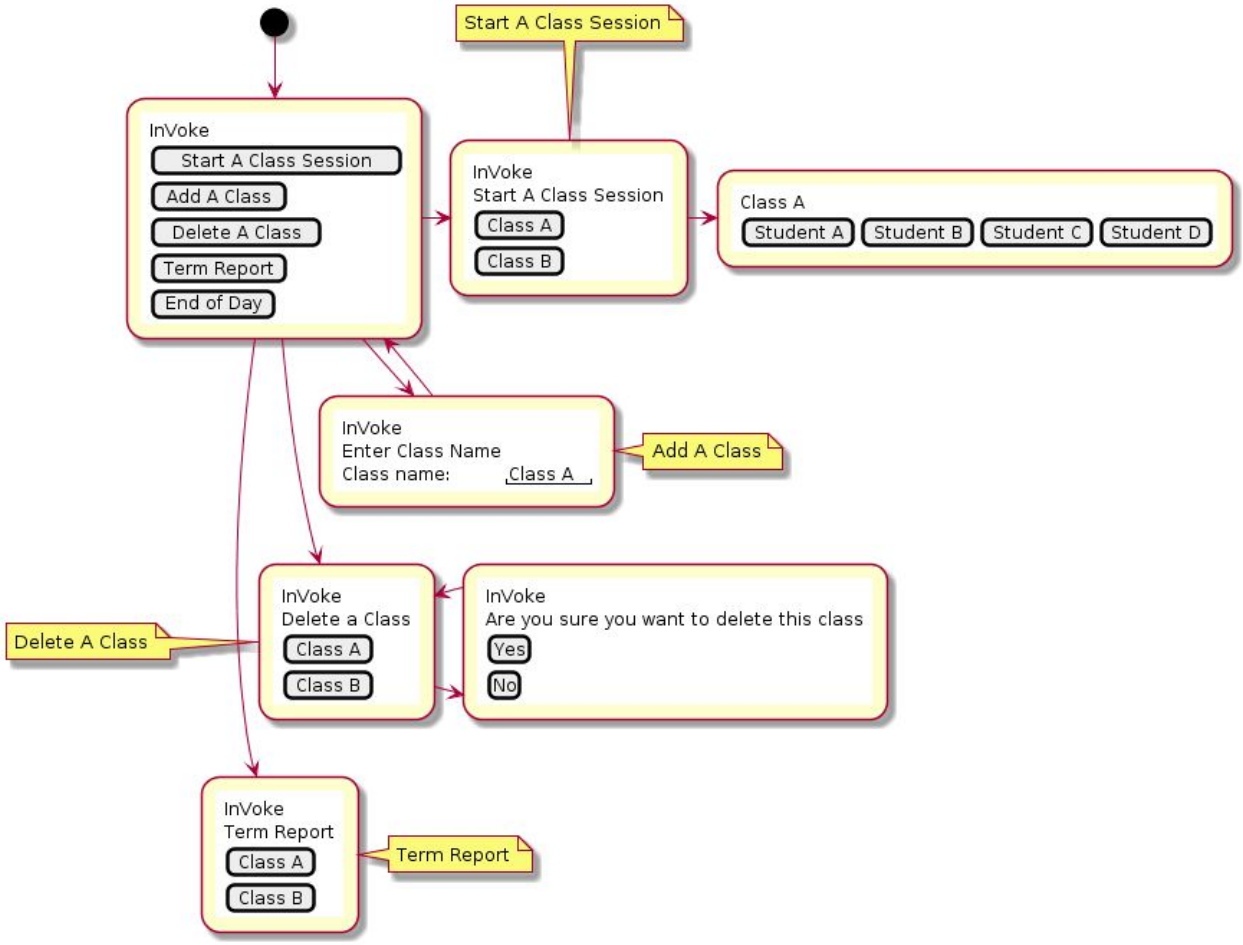


Figure 4.1 C2. A user interface diagram showing the menu layout of the user interface the instructor will use.

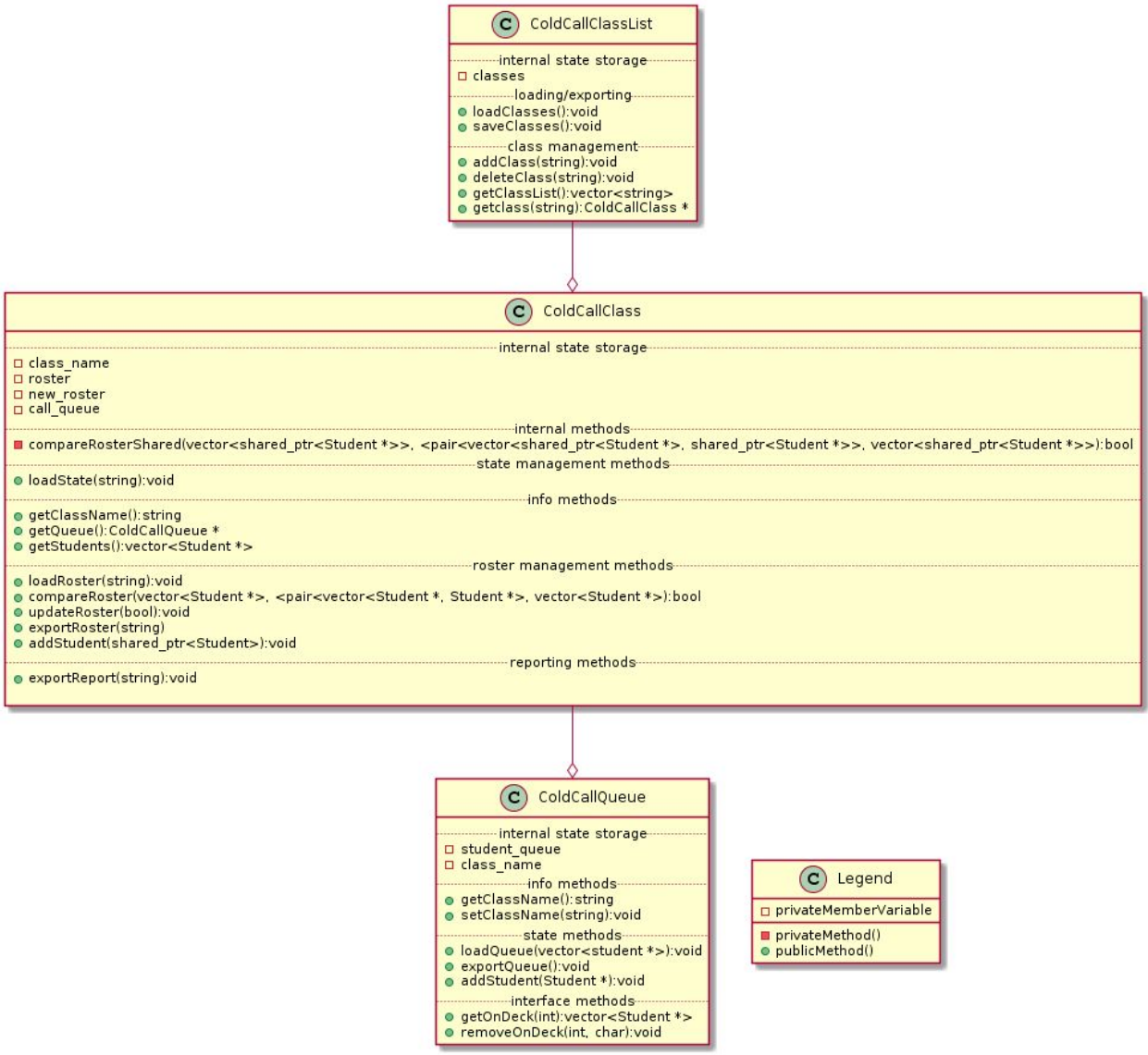
D. A design rationale.
   - The development of the Instructor Interface Module is critical to the usability of the system. It allows the user to work with classes and students easily and effectively.
   - The design rationale for this module is that there is a main window with a menu that allows access to the different functions of the program. The user only sees one thing at a time that is relevant to what they are currently doing. As an example, when using the cold call functionality, they only see the cold call list that shows a list of students "on deck." In another case, the user may want to view/update class rosters. During this task, they do not need to see the cold call queue, so that window is no longer visible.

E. Alternative designs.
   - We have been working on developing several user interface prototypes. During this process, we have discussed the addition of optional features such as having the ability to switch between classes in tabs. In addition to this, we have been covering the benefits and downsides of having separate windows for different

functionalities. These were unfortunately unable to be realized with the current code and.

4.2 - Cold Calling Data Module

A. The module's role and primary function.
    - The Cold Calling Data Module handles maintaining the roster and queue of students to be called upon during the program's runtime and acts as a bridge between the interface the instructor uses and data that is stored, processed and exported.
    - The Cold Calling Data Module contains a Cold Call Class List that contains the individual Cold Call Class instances and provides functions to load in previously used classes and functions to obtain the current classes.
    - The Cold Calling Class contains the roster of all the students for the class and provides functions to update the roster or export the current roster to a file. It also contains the Cold Calling Queue for that specific class.
    - The Cold Calling Queue contains a queue of Student objects and functions for accessing the "on deck" students and removing students from "on deck", logging their participation.
    - Student objects represent individual students in the class and their basic attributes
B. The interface to other modules.
    - Provides functions for the interface to get a list of classes currently in the system.
    - Provides functions for the interface to get a list of students who are in a class and to import a new roster of students, updating a current class.
    - Provides a function for the interface to get a list of students who are on deck for a specific class.
    - Provides a function for the interface to remove a student from on deck and the option to "flag" that removal.
    - Uses the data storage and reporting modules to store the current state of the classes and their students and to generate reports and logs of their participation activity.
    - Provides a method for the data storage module to load student information from a class roster or previously saved queue to be used for the current session.
C. Models



- Figure 4.2 C1. A UML class diagram showing the structure of the Data Module.

D. A design rationale.
   - The Cold Calling Data Module is essential to the overall system. We knew that it would be necessary for the program to store class data as well as information about the state of the cold call queue. In addition, we knew that we would need a module to modify and manipulate this data. The Cold Calling Data Module is our solution to this problem.
   - We decided to have this module connected in between both the Cold Call User Interface and the Data Storage/Reporting Module. This allows the program to have quick access to the data from both the front and back end of the system.

E. Alternative designs.
   - There are a few ongoing discussions about certain functionalities we have talked about implementing. One, for example, is to have an undo functionality. In order to implement this, the system will have to either wait to update the system or store information about the previous state to change back to it. This leads to some

concern about the scope of the project so we decided against it. The ability to undo accidentally removing someone from the 'on deck' state would be highly valuable to the end-user but it complicated storing the state data in a way that could be difficult to implement and troubleshoot.

4.3 - Data Storage/Reporting Module

A. The module's role and primary function.
   - The Data Storage/Reporting Module contains several functionalities that are important to the maintenance of the system. The goal of this module is to store the data between the program uses and to export the data in useful formats for the instructor.

B. The interface to other modules.
   - The Data Storage/Reporting Module communicates with the Cold Calling Data Module, which provides the data that is to be stored
   - The Cold Calling Data Module communicates with the Data Storage and Reporting Module to save and load state information, roster information, and reports.

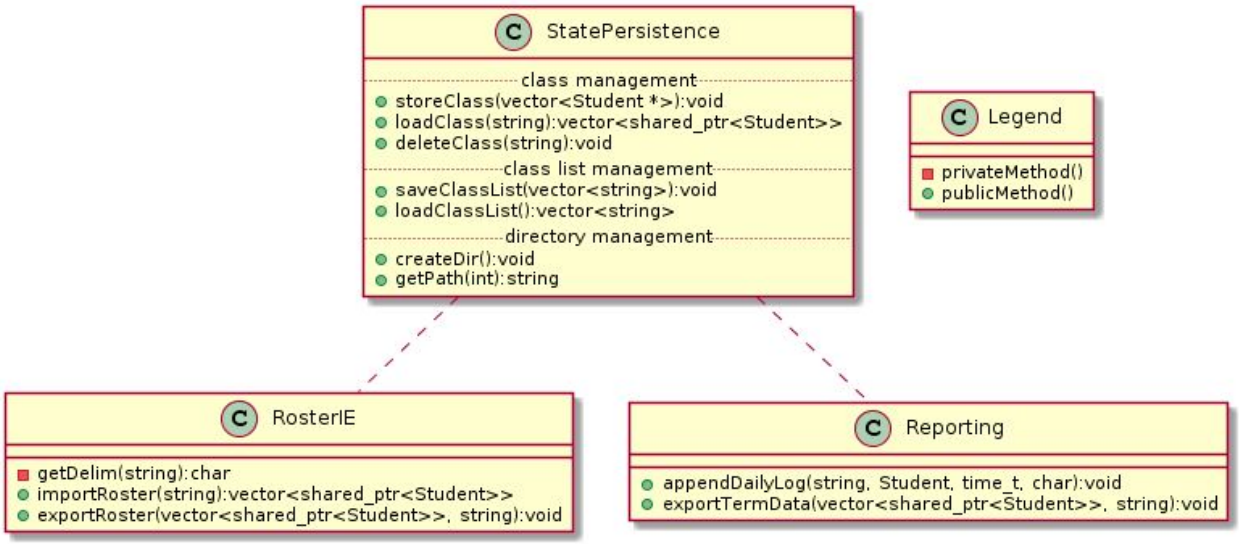
C. Models
   - 

     Figure 4.3 C1. A UML class diagram showing the structure of the Data Storage/Reporting Module.

D. A design rationale.
   - When designing our system, we realized that it would be of utmost importance to implement state persistence, so that the user could close the application and keep their current settings and class queue state. We also recognized that the user may want access to console logs, class rosters, and performance information. We developed this module to focus on and support the needs of the user.
   - This module communicates directly with the Cold Call Data Module. It does not, however, communicate directly with other methods within its own module. We

determined that these methods did not need to communicate with each other within the module as they are fairly independent in their functionality.

E. Alternative designs.
    - We discussed the possibility of having only one class available in the interface. Most instructors (the users of this software) will be teaching more than one class and more than likely want to use this program with more than one class. Supporting multiple classes was something we were able to support and made sense from a usability perspective.

5. Dynamic Models of Operational Scenarios (Use Cases)

Use Case A: Start of term roster loading

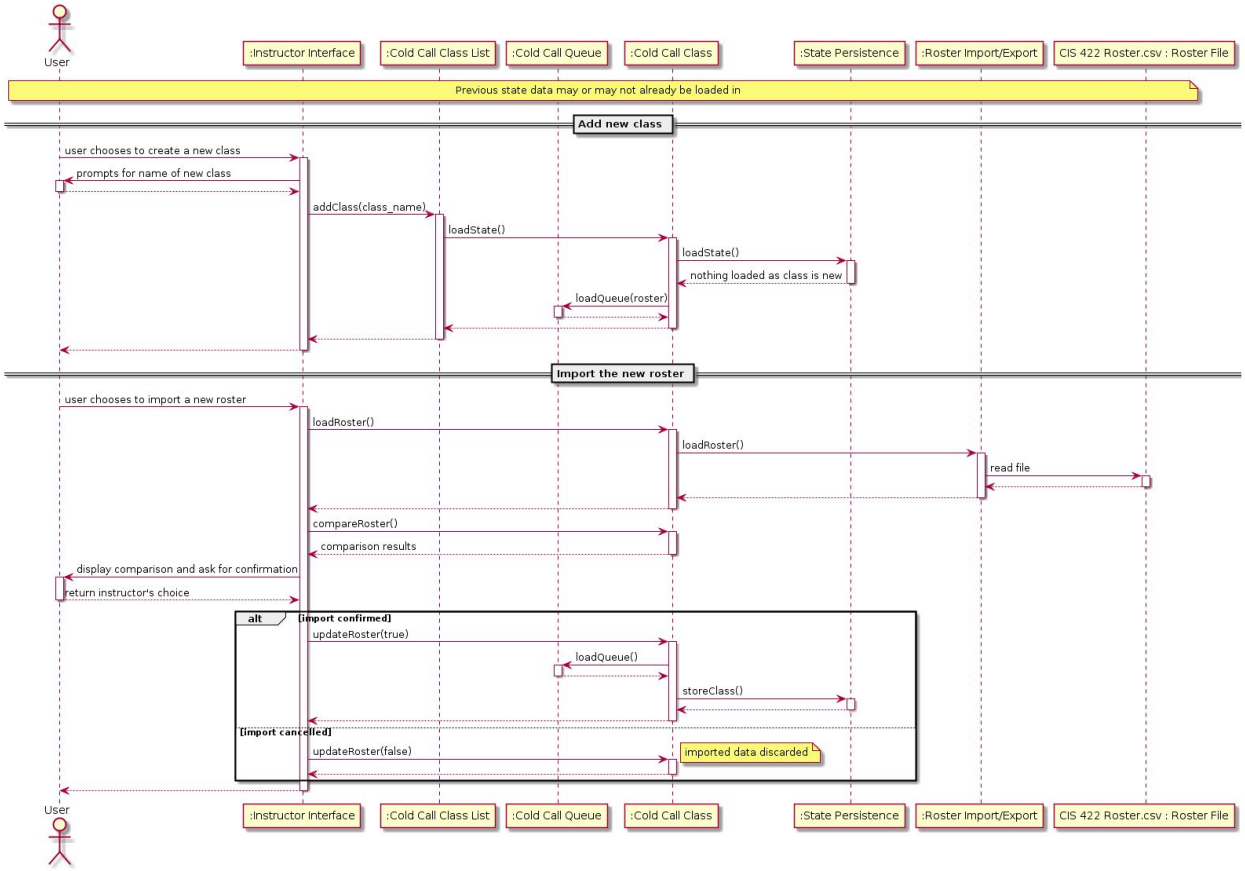


Figure 5.1. A UML sequence diagram showing the “Start of term roster loading” use case where a class is added and its roster is loaded in for the first time.

Use Case B: Loading previous state data

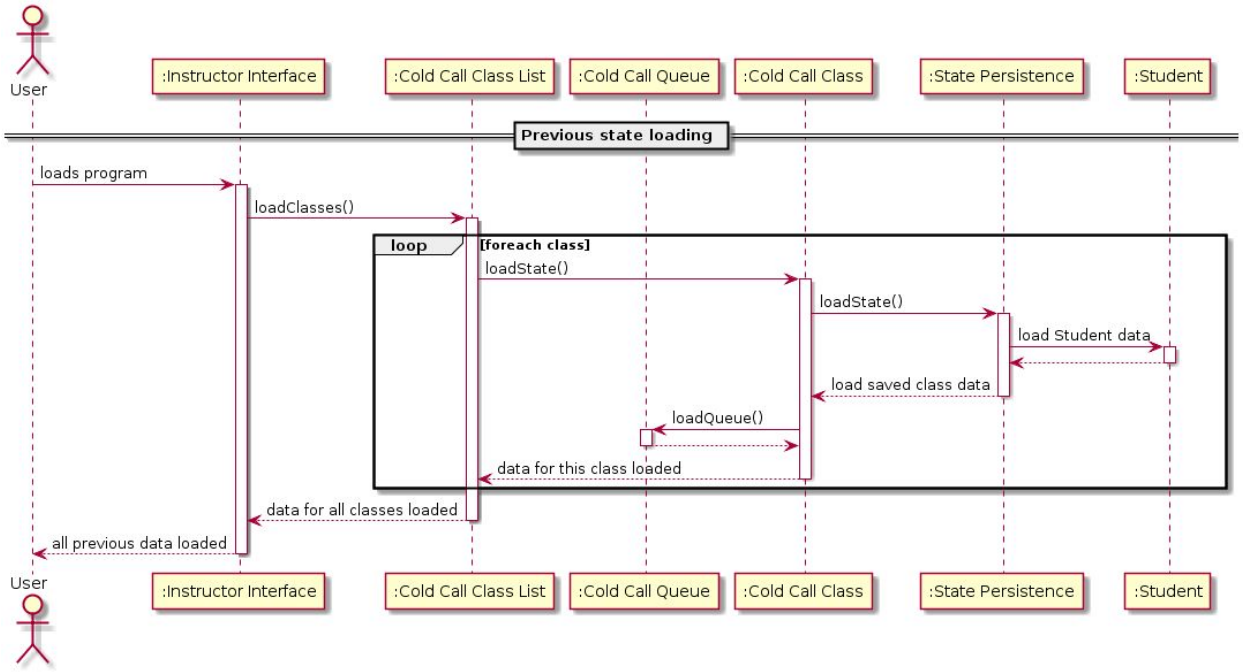


Figure 5.2. A UML sequence diagram showing the “Loading previous state data” use case where data from previous runs of the program is automatically loaded on startup.

Use Case C: Adjust class roster

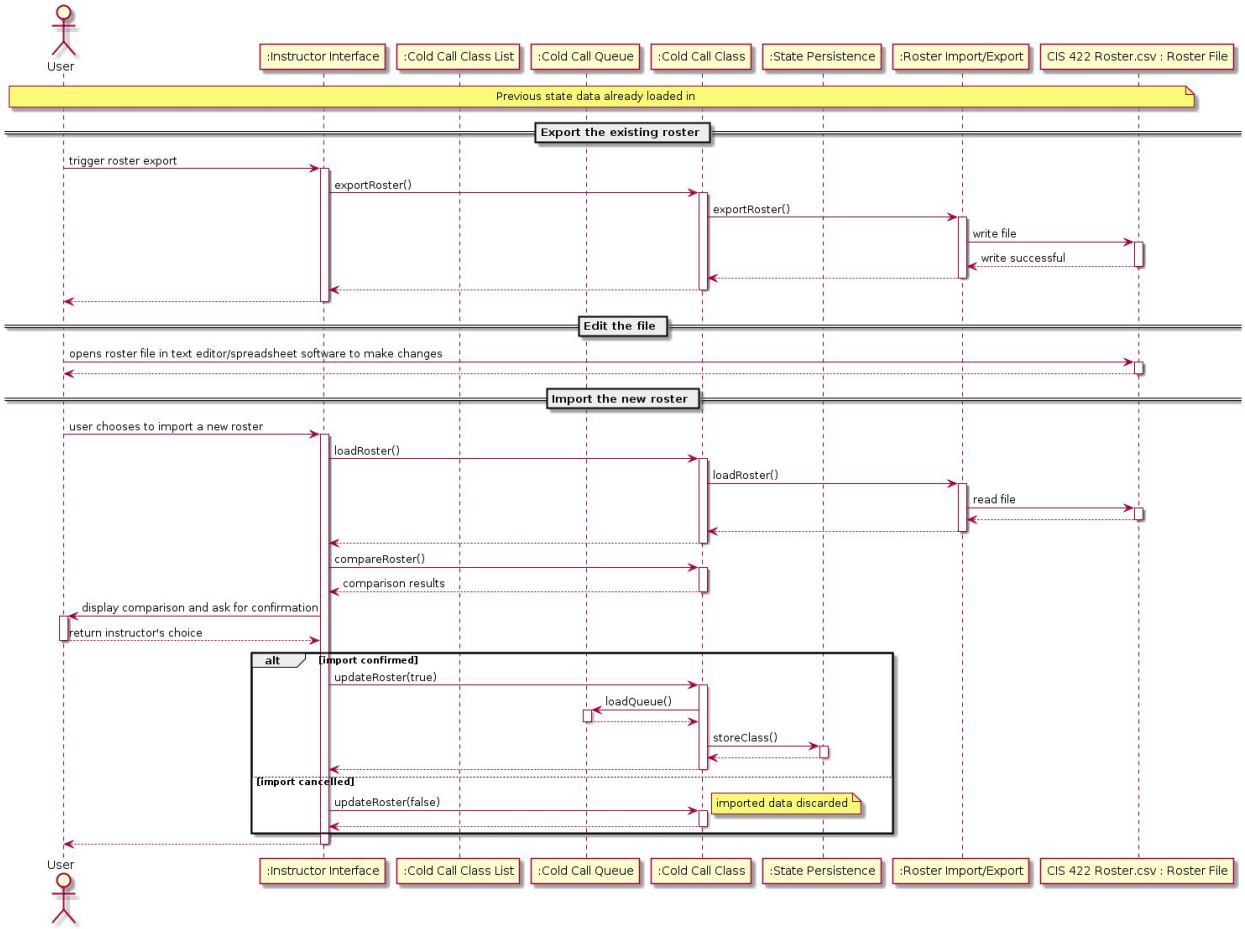


Figure 5.3. A UML sequence diagram showing the “Adjust class roster” use case where the current roster for a class is exported to a roster file, the user modifies the roster file, and the roster file is loaded back into the system after confirmation from the user.

Use Case D: Calling on a student

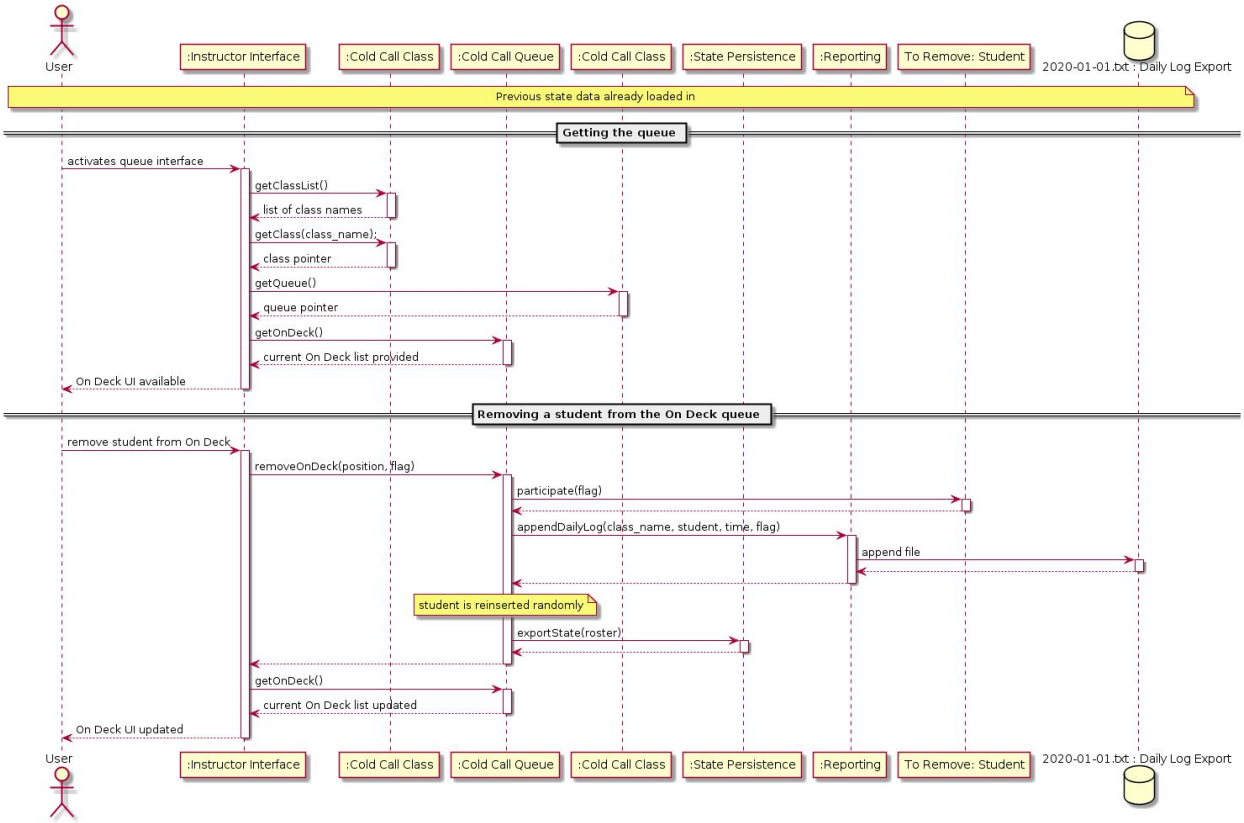


Figure 5.4. A UML sequence diagram showing the “Calling on a student” use case where an instructor pulls up the “on deck” students and calls on one of them.

Use Case E: After class review of daily log

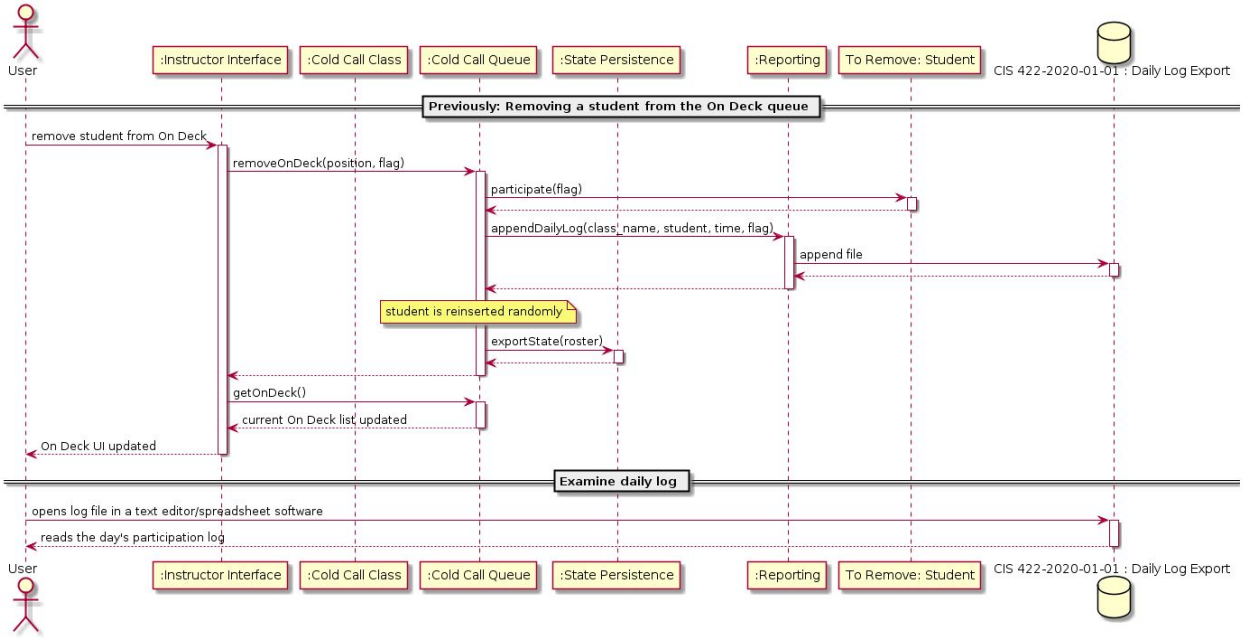


Figure 5.5. A UML sequence diagram showing the “After class review of daily log” use case where the daily log file was previously exported (shown in the first section) and is examined by the user in other software (shown in the second section). Note that the software is not used directly in the review of the previously exported log file.

Use Case F: After term review of summary data

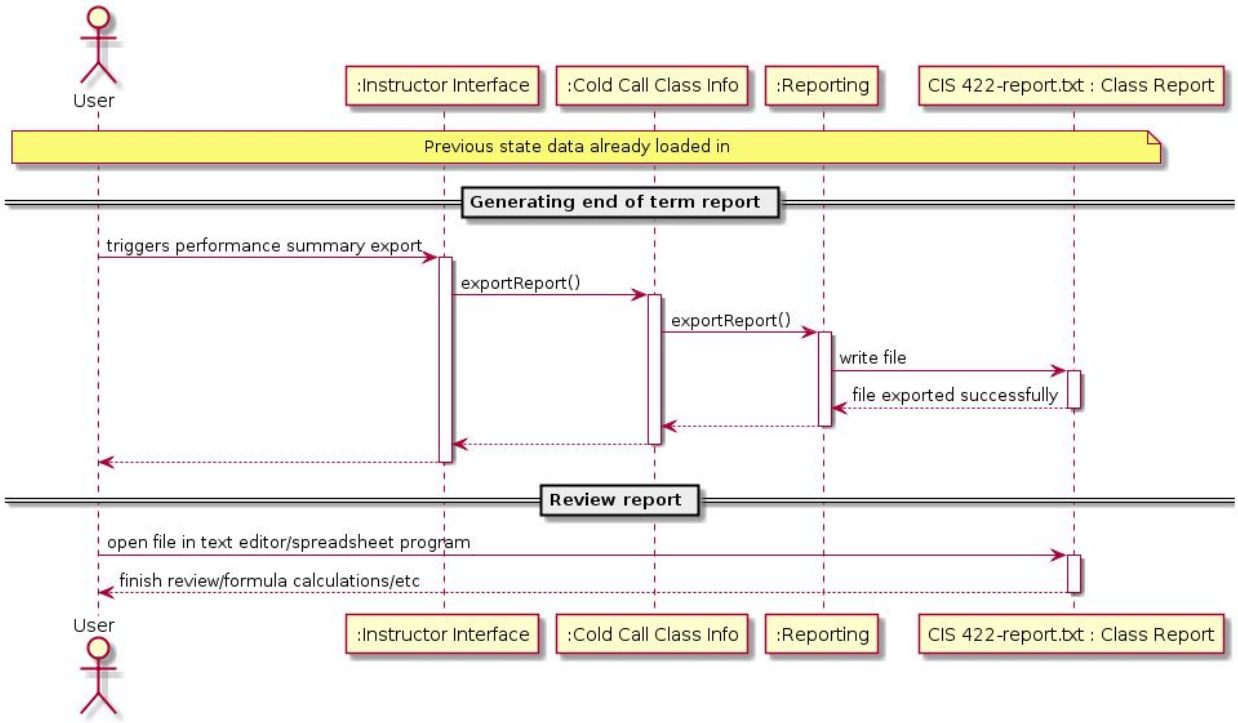


Figure 5.5. A UML sequence diagram showing the "After term review of summary data" use case where the triggers an export of a summary report (shown in the first section) and that report is examined by the user in other software (shown in the second section). Note that the software is not used directly in the review of the previously exported log file.

6. References

Hornof, Anthony. (2020). CIS 422 Document Template. Downloaded from https://classes.cs.uoregon.edu/20W/cis422/Templates.html in 2020.

Faulk, Stuart. (2011-2017). CIS 422 Document Template. Downloaded from https://uocis.assembla.com/spaces/cis-f17-template/wiki in 2018. It appears as if some of the material in this document was written by Michal Young.

7. Acknowledgments

This template builds slightly on a similar document produced by Stuart Faulk in 2017, and heavily on the publications cited within the document, such as IEEE Std 1016-2009.

PlantUML was used to generate the UML diagrams. See https://www.plantuml.com/ for more information about this software.