Zacree Carroll - Jan 28, 2022
Luke Vandecasteele - Jan 28, 2022
Nick Onofrei - Jan 28, 2022

Cold-Call Assist
Software Design Specification

1/27/2022

L.Vandecasteele(lv), Z.Carroll(zc), N.Onofrei(no), K.Nguyen(kn), H.Zhang(hz)

Table of Contents

1. SDS Revision History

Date	Author	Description
1/6/2022	no, kn, lv, hz, zc	Created first iteration of SDS, first work
1/11/2022	no, kn	Finished first iteration for initial submission
1/27/2022	zc, lv	Started revising initial SDS for finished product
1/28/2022	lv, zc, no	Continued work on section 4 and 5
1/29/2022	lv	Section 4 work

2. System Overview

The Cold-Call Assist software was created with the purpose of aiding teachers in a classroom setting with a proper method of cold-calling students [insert citation here for cold calling]. This software provides several functions including an interface for providing a set of “on-deck” students from which to call on, a database for storing a list of students, functionality to import student lists into the database as well as export student lists for changes to the student roster, and an algorithm designed to evenly distribute cold-calling amongst all the students and flag students in a summary file for after class interaction.

3. Software Architecture

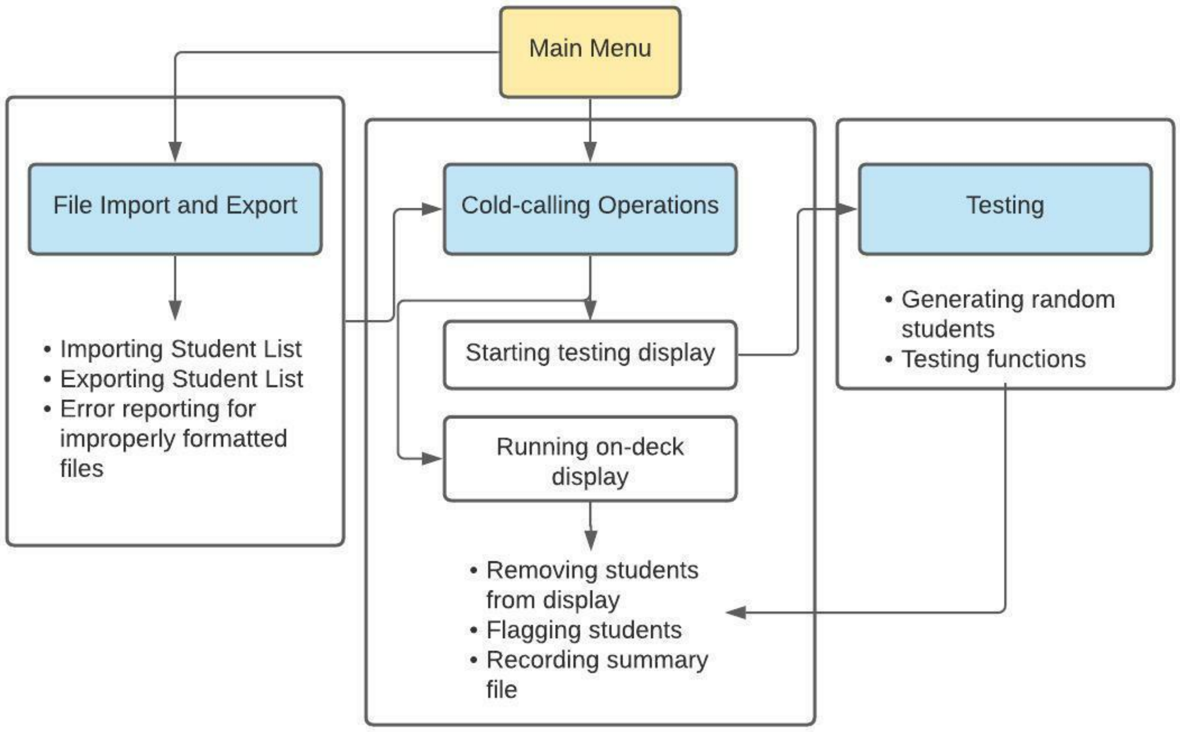

Figure 1: A Software Architectural diagram of Cold-Call Assist. Light blue indicates a software module with its functions grouped below the module and encompassed in a larger box. Light yellow indicates the launcher for starting the program and arrows show flow of information.

Main Menu

The Cold-Call Assist Software starts with the initial launcher for the program which connects the main software modules of the program. We can think of this as a main menu, or a system bus in computer hardware rather than a module as it provides a simple interface for easily accessing the main parts of the software. It stores no information about the current state of the program: it is just a UI.

Software modules

1. Cold-Call Operation: This module contains a majority of the logic for running and executing the Cold-Call assist software. The two main functions of this module include a display of on-deck students and a keyboard interface by which to start-up the testing software module of the system.

    a. On-Deck Display: This portion of the module includes the major functions that one considers when designing a cold-call system. This includes:
        i. functions for removing students from the on-deck display
        ii. functions for flagging students for a later follow-up
        iii. an algorithm for evenly distributing the number of times each student appears on on the display
        iv. Outputting student information to a summary file providing a detailed log of which students are cold-called in a given day.

    b. Testing Module Start-up: The second major action of the module is to provide a keyboard interaction for the user to launch the testing software module. This is a slightly hidden method by which the user (or professor) can run the program with a randomly generated data set to ensure that the installation of the software was successful.

2. Roster import and export: This module provides a set of functions to store student data and enable the user to alter student information and view summary files. More specifically these file manipulation tools that are a part of the module that imports a student roster file, check roster files for proper formatting, provide a method of displaying errors for improperly formatted files, storing student information in a database that is more readable for the Cold-Call Operation module, and exporting student roster files for the professor to alter and re-import. This module only interacts with the Cold-Call Operation module to provide data.

3. Testing: This module contains all the necessary testing functions and methods to run a test to see how the cold call system works. There are two parts to this.

a. <u>Generating Randomized Student Data And Storing It:</u> This portion generates random student ids, picks randomly from a bank of first and last names, puts the appropriate phonetic spelling to each name, and creates the email for each student that is then stored in a file.

b. <u>Running and Storing Test Procedure:</u> This portion randomly chooses students from the randomly generated student file and randomly chooses whether or not to flag the student. The testing procedure is supposed to reenact what 100 random cold calls would look like. The summary file is stored immediately after the testing procedure.

4. Software Modules

4.1 Cold Call Operations

Role and Primary Function

The Cold Call Operation module provides most of the driver functionality for the cold-call software. Some of the sections in the module include the FirstInterface(), the CCInterface(), AddAndRemoveStudent_1 (and 2, 3, and 4), Flag(), and Print_Summary_File(). These functions combined are able to operate the cold-call system completely independent provided there is information in the database containing student information. These functions will be described in more detail:

- FirstInterface(): This function uses tkinter to create a main menu user interface for the professor to navigate around the various functions of the software. It includes major buttons to import a student roster, export the existing student roster to make changes, and a button to launch the on-deck interface.
- CCInterface(): Launched via the FirstInterface(), this function shows the list of on-deck students that can be cold-called on. This function also binds certain keys to remove students as well as flag them for the professor to follow up with the student later. There are a total of 4 keybinds that can be altered by the professor that each correspond to a single student that is on-deck to remove them. Similarly there are 4 flagging keybinds that can be remapped and can be used to flag any of the 4 students within 1 second of them being removed.
- AddAndRemoveStudent_1(): This is a set of functions to respond to the 4 keybinds for removing a student. There are 4 functions, one for each keybind. These functions remove their corresponding student, update the on-deck display and spawn a thread to print the student that was removed and potentially flagged.

- Flag(): The flagging functionality is a single function of which 4 different keybinds can be pointed towards. When a student is removed from the on-deck display, any one of the 4 keys can be pressed within one second to flag the student in the summary file.
- Print_Summary_File(): After removing a student from the on-deck display this function is called via a new thread. The function first waits for 1 second to give the user time to potentially flag a student. Then it checks if the student was flagged and prints the appropriate line to the summary file depending upon if the student was flagged.

Interface Specification

The purpose of this module is to provide functionality for running the display of on-deck students and removing and flagging students. It interacts with the **File Import and Export** module to get the student list and check if there is a saved student list in the file. It also interacts with the **Testing** module to run a test of the functionality of the system.

A Static Diagram

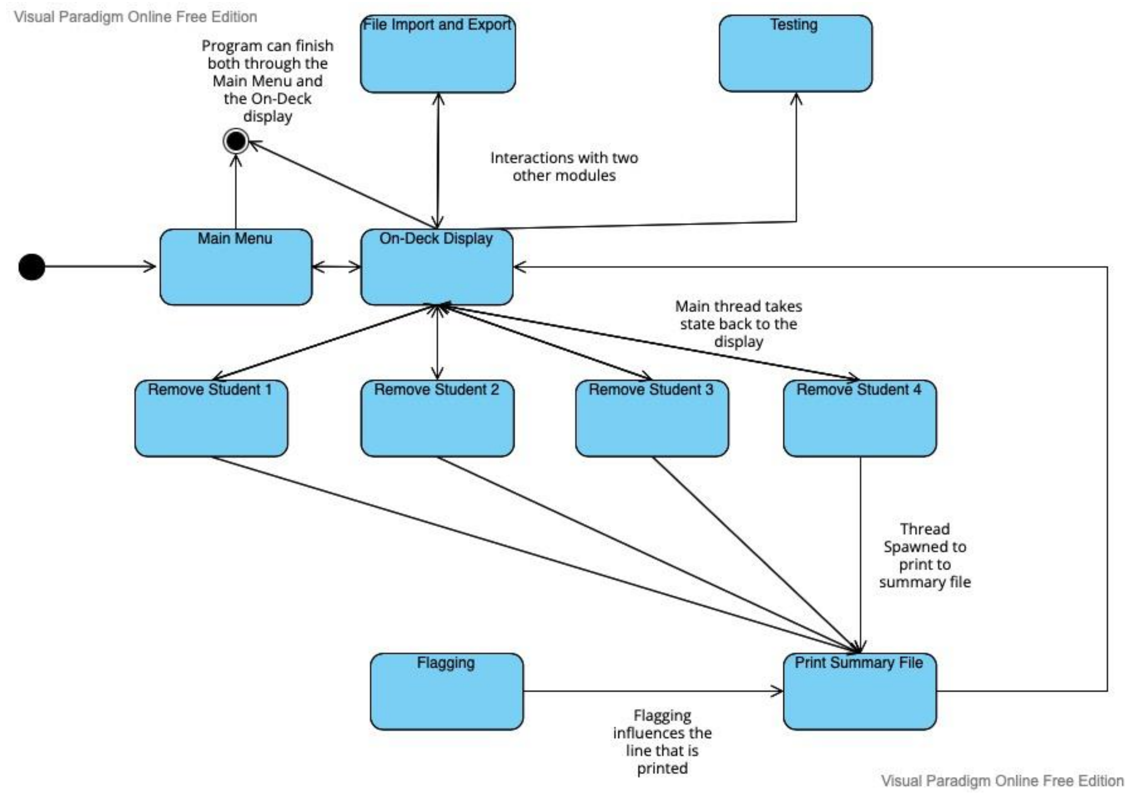
Visual Paradigm Online Free Edition
File Import and Export
Testing
Program can finish both through the Main Menu and the On-Deck display
Interactions with two other modules
Main Menu
On-Deck Display
Main thread takes state back to the display
Remove Student 1
Remove Student 2
Remove Student 3
Remove Student 4
Thread Spawned to print to summary file
Flagging
Print Summary File
Flagging influences the line that is printed
Visual Paradigm Online Free Edition

Figure 2: A UML state diagram for the Cold-Call Operation module. Includes interactions with the other modules but does not show a full state diagram for those modules.

Design Rational

The operations placed in this file are intended to group all the necessary functions for executing the most basic components of a cold call system. This means that if the system were to already have a set of data saved in it, the Cold-Call Operations module can exist entirely on its own and needs none of the other modules. This can not be said for the other modules since although they provide essential functions towards the cold-call software as a whole, they cannot exist independently. The main purpose of grouping all these functions together was to make it easy for future code maintenance. If all the essential cold-calling operations are in the same module (and consequently the same .py file) then it simplifies any future changes that need to be made to the system.

Alternative Designs

One additional way we originally had our code structured was to separate the firstInterface() functionality into a main.py file. The reason we did this was because we intended for this to be the driver function for the entire software. However, since the main menu of the firstInterface() function did not serve any purpose other than to provide an interface to navigate to the other aspects of the software. It contained no actual other functions other than the main menu so we thought that it was redundant to keep it in its own file.

A second design consideration we made was how to print the summary file. Because the summary file function needs to wait to see if a flag was detected before proceeding, and we didn't want our entire system to pause while this was happening, we decided to use some simple multi-threading. That way the main thread could continue on with the on-deck interface while the additional thread could print to the summary file. This was something we changed fairly recently since we had some inconsistencies with the flag reporting in the summary file.

Another consideration, although brief, was to switch to the arrow key method listed in the SRS because we were struggling with detecting the flagging within one second. However, we were able to come up with a solution to this without much fuss.

Finally, one other major alternative design change we made was to modify how we queued students for the on-deck display. At first, we had a standard queue. However, we felt this wasn't specific enough since we did not like that it didn't keep track of the number of times each student was flagged. Next, we decided to use a priority queue so that we could ensure that students with the fewest number of cold calls would have a higher priority and thus be selected first for the on-deck display. This had a major drawback. Since we placed the students in a queue using a tuple where the first entry was the number of cold calls and the second was the students name the built in python priority queue would first place students based off their priority, and then according to their name. This meant that students would end up alphabetically placed in the queue if they had the same priority, which removed the random selection process for on-deck students. To remedy this we first created a class Student which was used more as a placeholder for the students name and priority. Next, we used operator overloading on the “<“ comparator of the class so that it would only compare the students priority for the “<“ and not consider their name. That way, when placed in the priority queue as a Student class they would first go off of priority, and then off the strict order that they enter the queue. This was our final design solution since it incorporated both the randomness and ensured that the cold calling was evenly distributed amongst all the students.

4.2 File Import and Export

- The role of this module is to provide support for the import and export of student roster files

- There are four main components of the file import and export module:

    - FileError() is used to create a visual display (using the tkinter framework) which shows what went wrong when attempting to import or export a file. It spawns a window setting the window’s title to the first argument of the function. It then attaches to that window a button which explains in more detail the nature of the error encountered.

    - ScanFile() scans a file for correct formatting and returns a list of lists if the second argument is set as True. This list of lists contains one entry list for each student and contains all of their information which was found in the roster file. If there was an error it calls the FileError() function.

    - ImportFile() opens a file browsing window (using tkinter) and waits for the user to select the file they wish to import. ImportFile() then scans the file for errors using ScanFile() with False as the second argument (because no student list is needed in that case).

- The file import and export module provides the previously listed functions to the cold call operation module only. Cold call makes use of ScanFile() after the file has been imported using ImportFile() in order to build the on-deck display to the user.

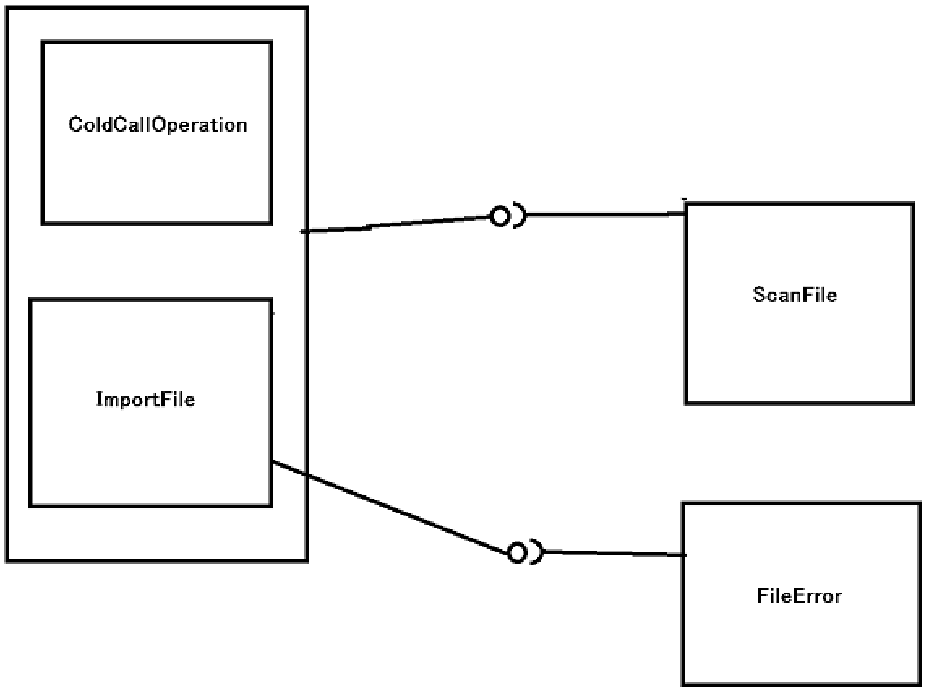

Figure 3: UML component diagram for the current file import and export module

Reselman, B. (2021, November 3). How to create static diagrams in UML. Retrieved from https://www.redhat.com/architect/static-UML-diagramming#object

- The design rationale for the import and export file module is centered around keeping similar functionality grouped together in the same module.
  - Tkinter is a graphical user interface framework which is being used in this project and it does not work well in certain instances when attempting to return values from functions which are used to provide functionality to button presses. So the file import export module will implement its own error handling via spawning tkinter windows and prompting the user for input in the FileError() function.
  - ScanFile() serves two purposes; ensure correct file format and return student information lists. It can be used for either desired functionality. This reduces the duplication of code. Allowing either use case creates an interface for the cold call module to make use of ScanFile() even when not importing a file.
- Alternative Designs:
  - ImportFile() initially returned information on the status of the file imported to the cold call operation module but this did not work well with tkinter.

This was changed so that the import and export file module contained its own interface to tkinter for file errors located in FileError().

4.3 Testing

- The role of this module is to create a testing procedure to see how the cold call system works with random generated student data.

- There are 4 main components. The first 3 are located in studentgen.py and the last one is located in viewAndController.py

    - studentfile() generates a file called StudentFile.txt and writes in it 100 random students. That includes their first name, last name, phonetic spelling, student id, uoregon email and reveal code.

    - caller() is the function that does all the random cold calling and random flagging as well. It takes each cold call pick and writes the result to the output file testoutputfile.txt. This function calls on studentfile() to generate the StudentFile.txt.

    - testwarningInterface() generates the tkinter window to warn the user that if they continue the previous stored information in testoutputfile.txt will be overwritten. This function calls on caller()

    - testflagging() checks to see if the test key has been pressed twice to initiate the testing procedure by calling on testwarningInterface() in studentgen.py

- The testing module works in correlation with the cold call operation module. At any instance when removing names the user can decide to test the system. The cold call operation module calls on functions in the testing module in order to compute the random cold call procedure.

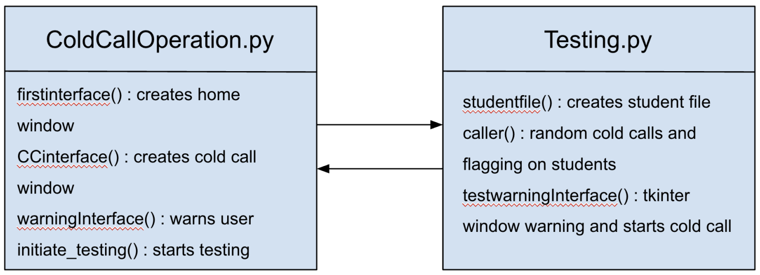
ColdCallOperation.py
firstinterface() : creates home window
CCinterface() : creates cold call window
warningInterface() : warns user
initiate_testing() : starts testing
Testing.py
studentfile() : creates student file
caller() : random cold calls and flagging on students
testwarningInterface() : tkinter window warning and starts cold call

Figure 4: UML Class diagram for the way the testing module works with the cold call operation module

- Design Rationale
    - The testing module was intended to keep all the testing in one file and connect it to the main window when the cold call is running. We decided on a simple combination (press the t key twice or whatever the user changes it to twice) to not make it too complicated for the user to test the system. A design option or concern was where to allow the user to test the system. We concluded that it was best to test the system when you are actually using the system. This is when you are at the on deck portion of the program. If you notice an abnormality in the program then to run the test the user would typically be at the cold call window.
- Alternative Designs:
    - One option was to create a button in the main window when first starting up the system. The reason we didn't do this was because it would make the interface more clumsy by using up more screen real estate. The system requirements also stated that it would like a secret key combination of some sort in order to test the system.

5. Dynamic Models of Operational Scenarios (Use Cases)

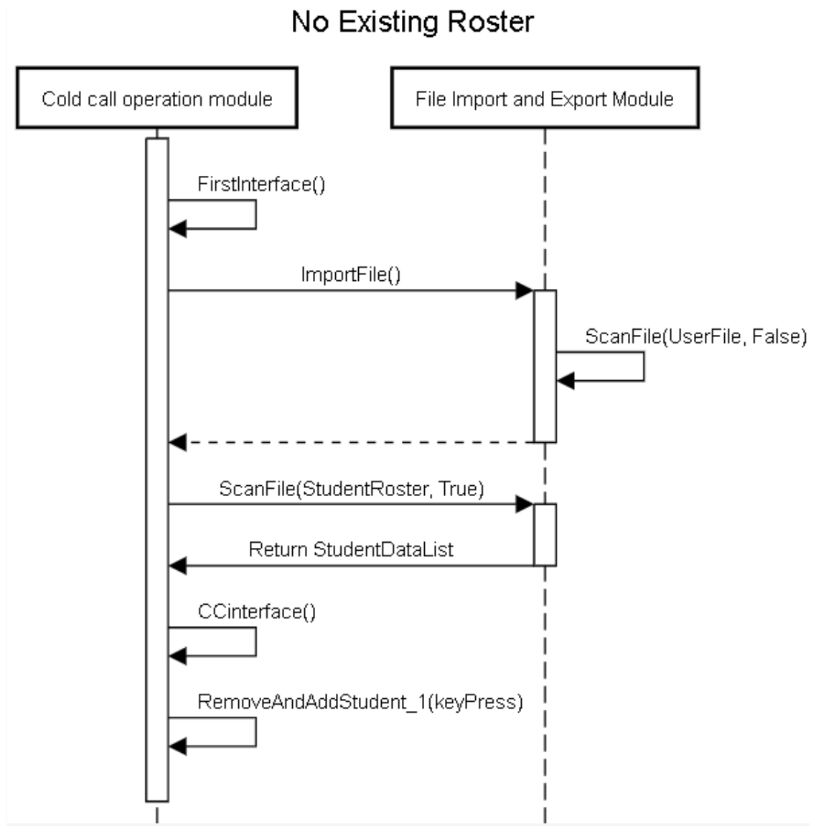
No Existing Roster
Cold call operation module
File Import and Export Module
FirstInterface()
ImportFile()
ScanFile(UserFile, False)
ScanFile(StudentRoster, True)
Return StudentDataList
CCinterface()
RemoveAndAddStudent_1(keyPress)

Figure 5: UML sequence diagram displaying initial start use case

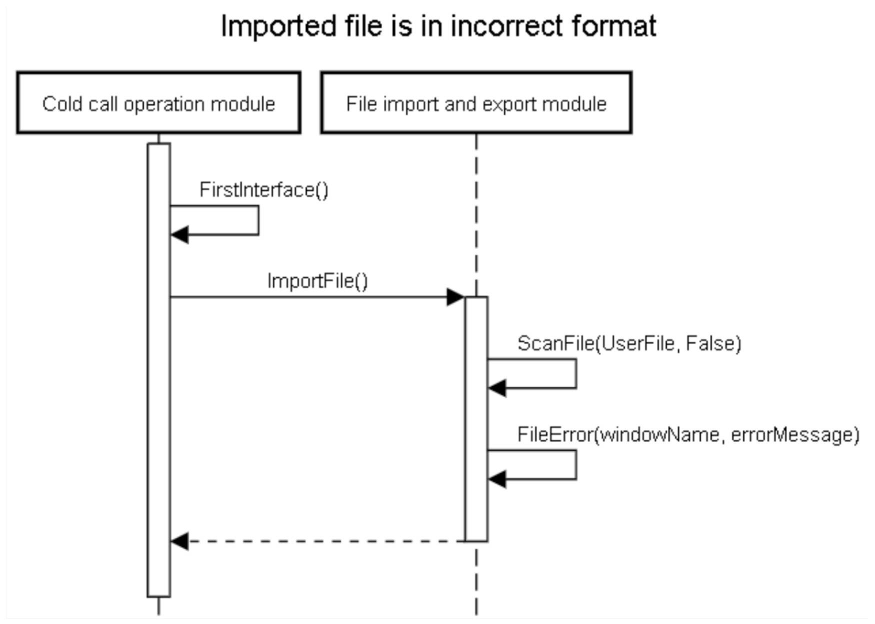
Imported file is in incorrect format
Cold call operation module
File import and export module
FirstInterface()
ImportFile()
ScanFile(UserFile, False)
FileError(windowName, errorMessage)

Figure 6: UML sequence diagram displaying sequence of events when the imported file is in the incorrect format

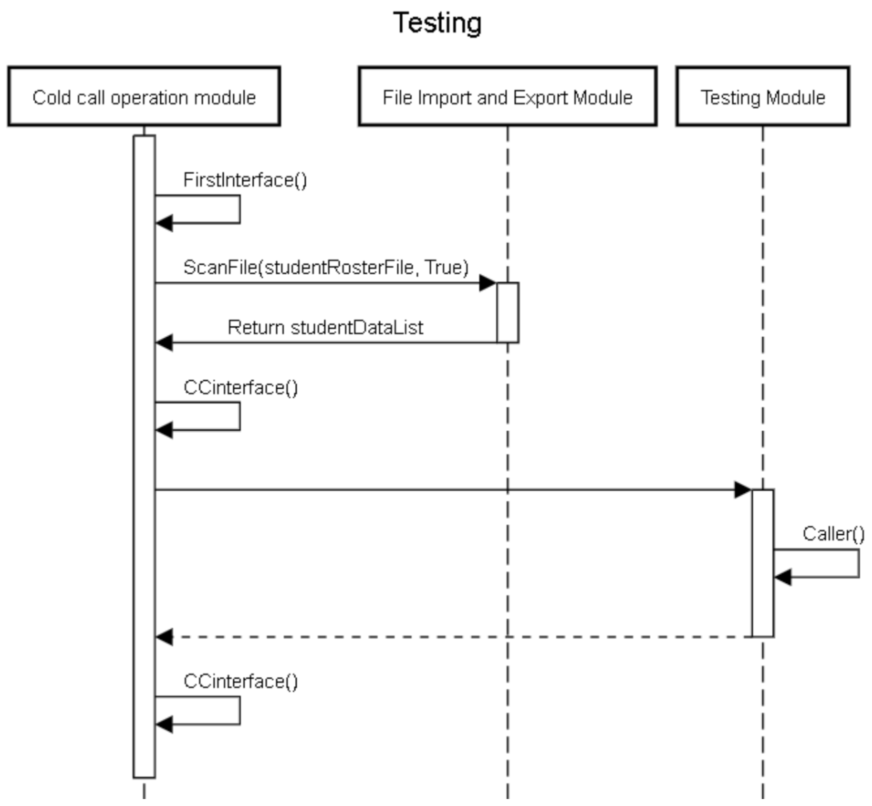

Figure 7: UML sequence diagram displaying sequence of events which take place during testing

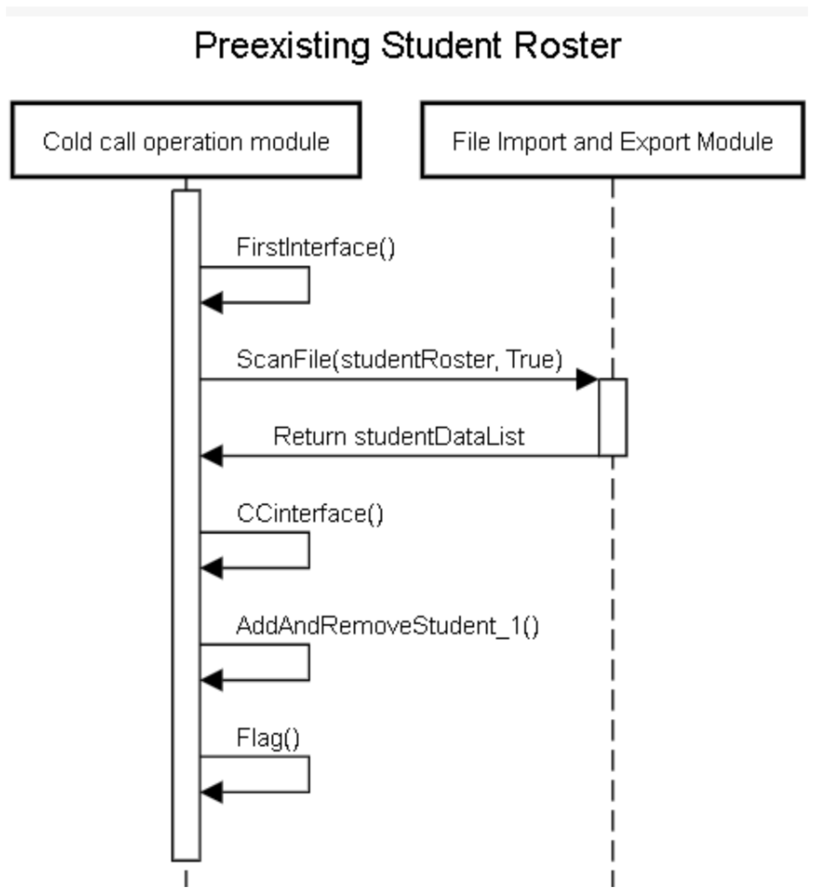
Preexisting Student Roster
Cold call operation module
File Import and Export Module
FirstInterface()
ScanFile(studentRoster, True)
Return studentDataList
CCinterface()
AddAndRemoveStudent_1()
Flag()

Figure 8: UML sequence diagram displaying sequence of events which take place during an average use of the system

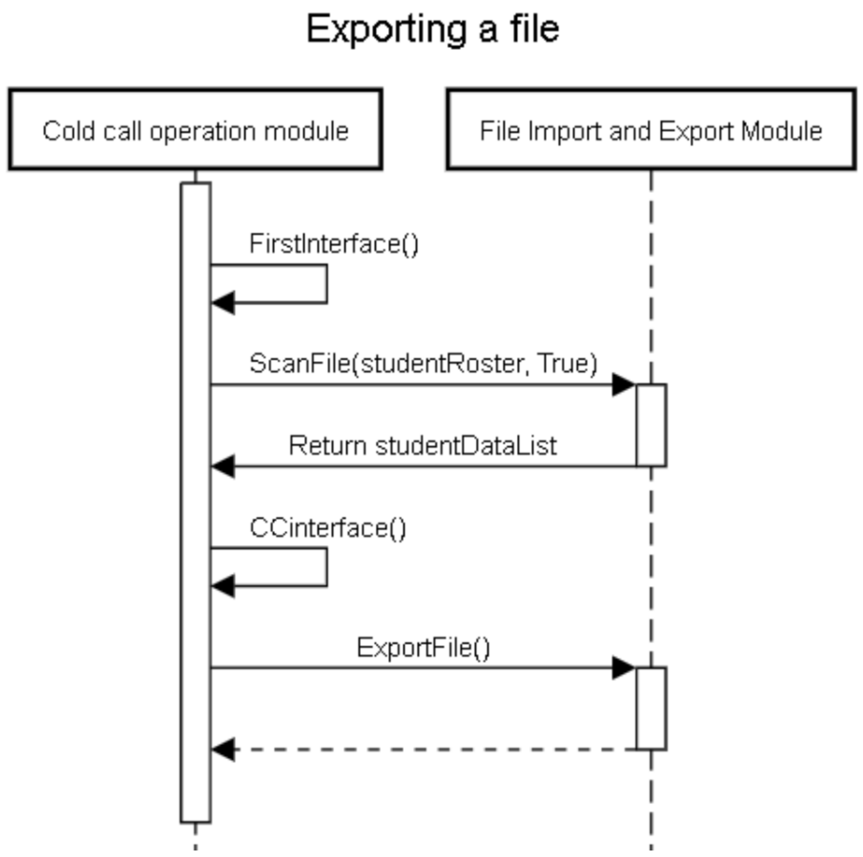
Exporting a file
Cold call operation module
File Import and Export Module
FirstInterface()
ScanFile(studentRoster, True)
Return studentDataList
CCinterface()
ExportFile()

Figure 9: UML sequence diagram showing the sequence of events that take place when exporting a file