# CIS 422 Project 1:
# Classroom Cold-Call Assist Software
# Software Design Specification

Mikayla Campbell (mc), Joseph Goh (jg), Olivia Pannell (op), Bethany Van Meter (bvm), and Ben Verney (bv)

February 3rd, 2020 - v2.0

## Table of Contents

# 1. SDS Revision History

| Date | Author | Description |
|---|---|---|
| 1-14-2020 | mc | Created the initial document. |
| 1-14-2020 | mc | Started writing sections for v1.0 |
| 1-16-2020 | op | Finished section 4 |
| 1-16-2020 | mc/op | Finished SDS version v1.0 |
| 1-17-2020 | jg | Formatting updates and minor edits |
| 2-1-2020 | mc | Started writing the second draft. |
| 2-2-2020 | op | Help with design rationale/peer review |
| 2-3-2020 | mc | Finished SDS version v2.0 |

# 2. System Overview

## 2.1. Description of System

This system will provide an instructor with the basic capabilities of "cold calling." "Cold calling" is a method in which an instructor calls on a student who hasn't recently participated in the class's discussion. This software will also provide the students with a "warm-up period." This is established through a queue that displays the top 4 students that may be "cold called" next. The instructor will also have the capability to remove students from the queue when they have participated in the class discussion and to flag students when they want to contact them after class. The main goal of this system is to "cold call" on students efficiently to improve class discussion and participation.

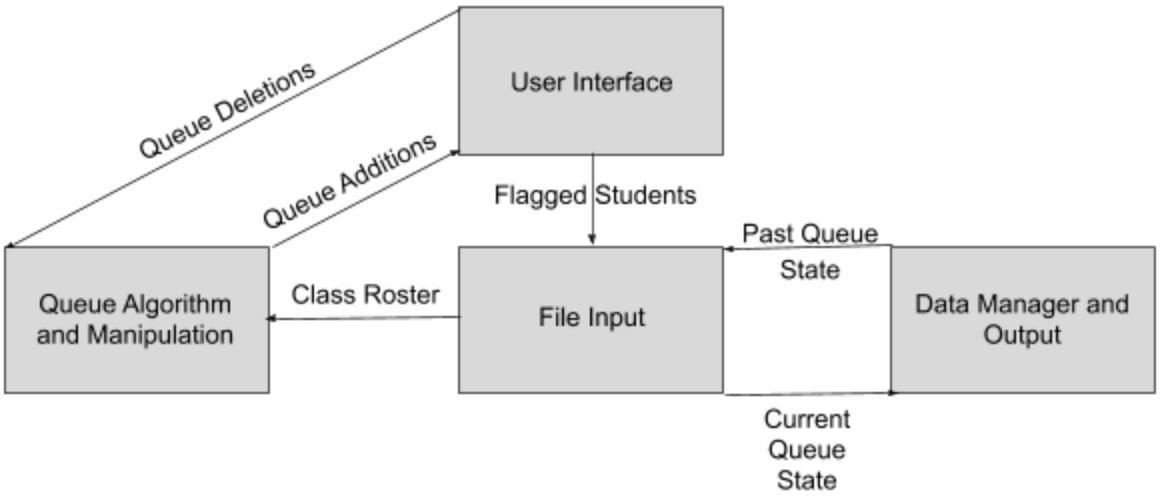
## 2.2. System Organization and Interaction

This system has the main functionality of "cold calling" on students in a classroom setting. This functionality is divided into a total of 3 main files:

1. ***ui.py*:** This file connects the entire program (Much like the functionality of a main file) and controls the program's graphics.
2. ***create_queue.py*:** This file contains the algorithm that sorts the students in the class into a queue that is used to cold call the students. This file also handles the manipulation (Deletion and addition of a student) of the queue. Also, this file reads and parses the class csv files, formats the data before it is saved, and reads in the saved data when the program resumes.
3. ***save_data.py*:** This file saves the state of the class's queue, saves the state of the class's sorted roster list, creates the class's daily log, and adds to the class's term summary when the program is either paused or exited.

# 3. Software Architecture

This system was created with one main functionality: “cold calling.” This functionality was implemented using the following architecture:




(Figure 3.1) “Cold Call” System Architecture

The User Interface Module is the module that acts as the main file of the program and contains the graphics of the program. The Queue Algorithm and Manipulation Module is the module that handles the creation and manipulation of the student “cold calling” queue data structure. The File Input Module is the module handles the reading, parsing, and formatting of imported data such as class files. The Data Manager and Output Module is the module that saves the system’s data and creates the resume program functionality. This software architecture is split into these modules to establish strong cohesion between the components of each module. This architecture also ensures that no two modules are strongly coupled. Most of the system’s modules are loosely coupled, with a stronger coupling between the Queue Algorithm and Manipulation Module and the User Interface Module and between the File Input Module and the Data Manager Output Module.

# 4. Software Modules

## 4.1. User Interface Module

***The module’s role and primary function.***

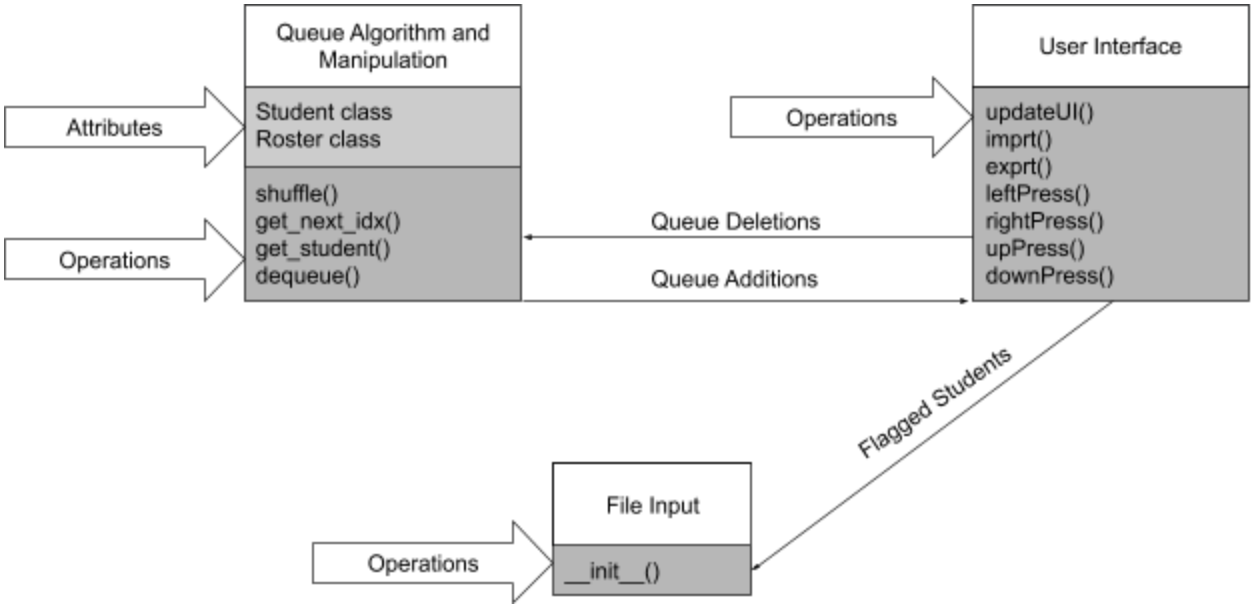
The User Interface Module’s primary purpose is to act as the main file of the program and tie the individual functionalities of this system into a single file. It also holds the design and

functionalities of the user interface graphics. This module contains the main program, the graphics, and the user controls.

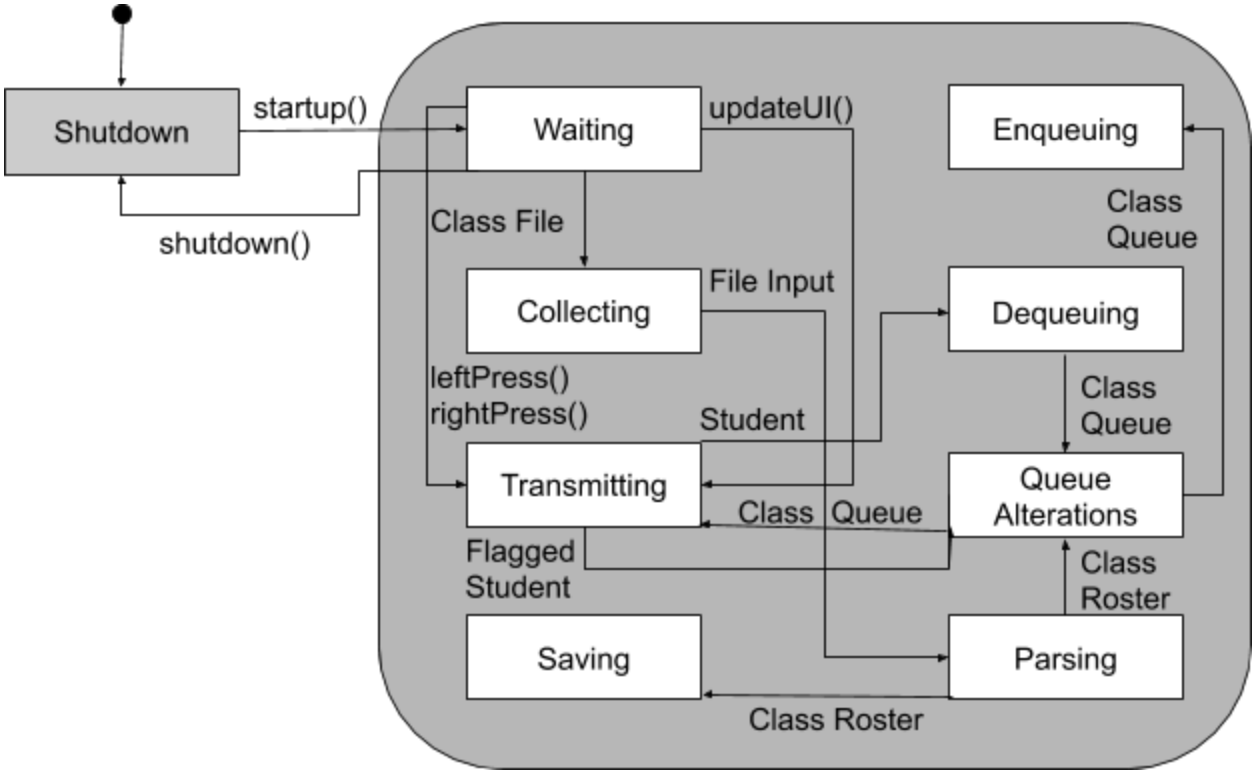
***Interface Specification***

This module interacts with the following modules: the Queue Algorithm and Manipulation Module, and the File Input Module. The User Interface Module receives queue alterations such as enqueuing a student from the queue from the Queue Algorithm and Manipulation Module, and the Queue Algorithm and Manipulation Module receives queue alterations such as dequeuing a student from the queue from the User Interface Module. The User Interface Module also inherits the Student class and the Roster class from the Queue Algorithm and Manipulation Module. The User Interface Module sends the flagged students data to the File Input Module.

***A Static and Dynamic Model***




(Figure 4.1.1) User Interface Module UML Class Static Diagram




(Figure 4.1.2) User Interface Module UML State Dynamic Diagram

***Design Rationale***

This module is the initial module that brings everything together. It is made to bring subsystems together to create a working system. It includes the graphical user interface of our project which connects to the queue. This module was decided to have weak cohesion with queue algorithm and manipulation module since it relies partly on having sections of the queue viewable and with the ability to dequeue those parts as well as weak cohesion with File Input module since it sends messages when a student is flagged.

## 4.2. Queue Algorithm and Manipulation Module

***The module's role and primary function.***

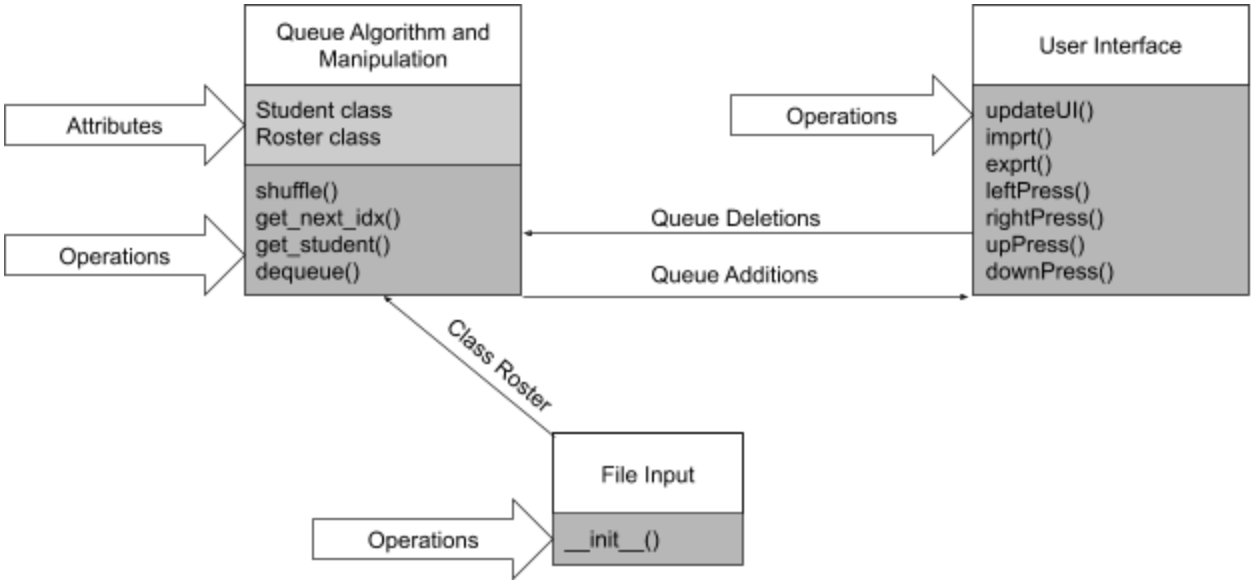
The Queue Algorithm and Manipulation Module's primary purpose is to handle the creation of the student queue, the manipulation of the student queue, and the randomization of the student queue. This module contains everything related to class "cold calling" queue.

***Interface Specification***

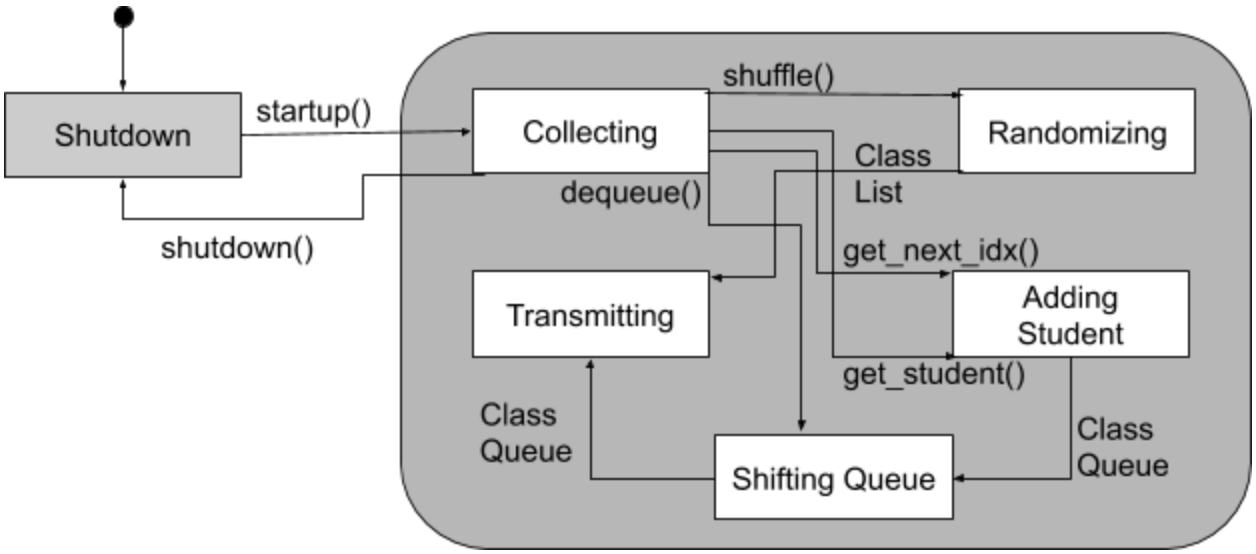
This module interacts with the following modules: the User Interface Module, and the File Input Module. The User Interface Module receives queue alterations such as enqueuing a student from the queue from the Queue Algorithm and Manipulation Module, and the Queue Algorithm and Manipulation Module receives queue alterations such as dequeuing a student from the queue

from the User Interface Module. The Queue Algorithm and Manipulation Module also receives the parsed class list from the File Input Module.

### *A Static and Dynamic Model*




(Figure 4.2.1) Queue Algorithm and Manipulation Module UML Class Static Diagram




(Figure 4.2.2) Queue Algorithm and Manipulation Module UML State Dynamic Diagram

### *Design Rationale*

The Queue Algorithm and Manipulation Module is designed to control the queue of the system. This module has weak cohesion and is loosely coupled with the two other modules it communicates with. It was decided that the modules it interacts with, File Input and User Interface Module, would let this module know when it was time to dequeue or enqueue. Because

this dequeueing and enqueuing is done inside this module it is mostly unrelated to the other modules it interacts with, thus, it needs to be its own module.

### 4.3. File Input Module

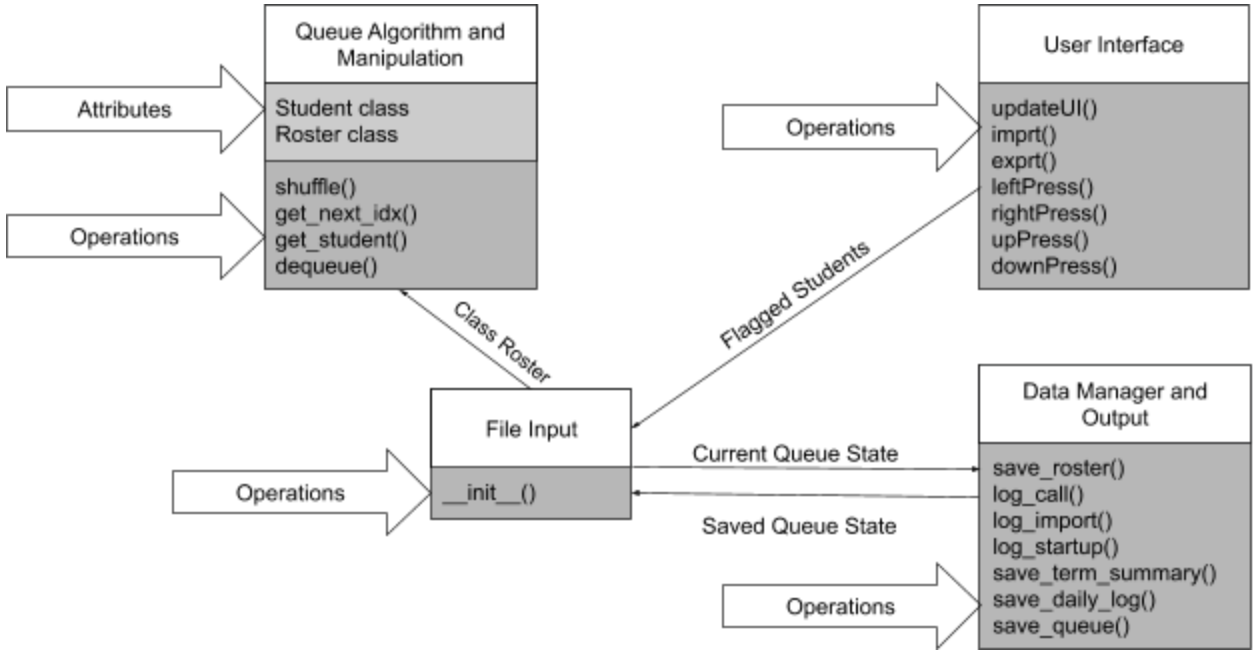
***The module's role and primary function.***

The File Input Module's primary purpose is to read in class files and format them to work with the structure of the class queue. This module contains the file parsing and importing of class files.

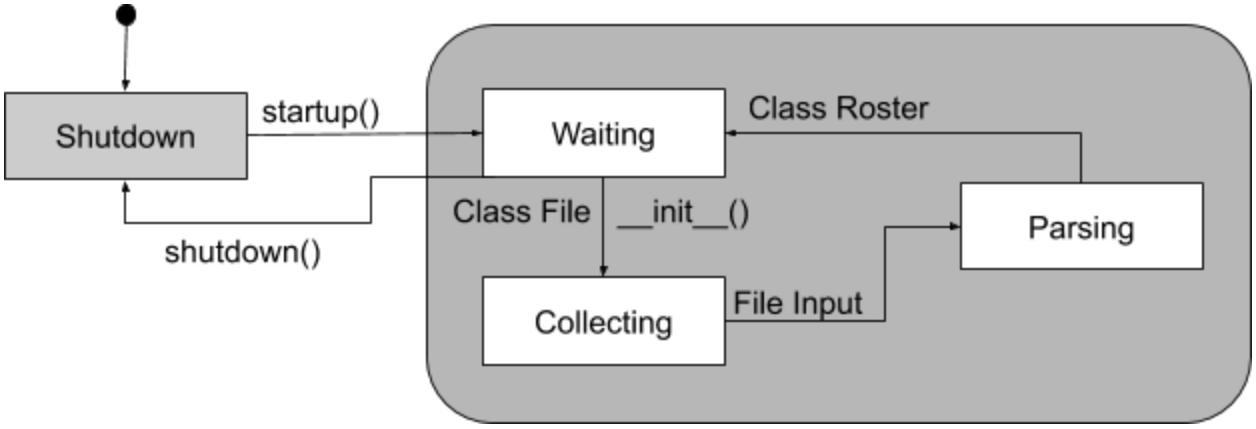
***Interface Specification***

This module interacts with the following modules: the Queue Algorithm and Manipulation Module, the User Interface Module, and the Data Manager and Output Module. The File Input module sends the formatted class roster to the Queue Algorithm and Manipulation Module. The User Interface Module sends the flagged students data to the File Input Module. The File Input Module sends the current queue state to the Data Manager and Output Module and the File Input Module receives the saved queue state from the Data Manager and Output Module. Lastly, the User Interface Module sends the flagged students data to the File Input module.

***A Static and Dynamic Mod***e




(Figure 4.3.1) FIle Input Module UML Class Static Diagram




(Figure 4.3.2) File Input Module UML State Dynamic Diagram

***Design Rationale***

This module is designed to control the input of the system. It makes sure the imported files are formatted properly and can be parsed into a class roster and eventually transformed into the queue structure. This functionality was made into its own module to keep the incoming program traffic separate from the rest of the program. This leads to less errors and makes the formatting of the files into class lists and queues smoother. This module also has weak cohesion and is loosely coupled with the other modules. The User Interface Module focuses on the graphics and controls of the system, the Queue Algorithm and Manipulation Module focuses on the queue data structure, and the Data Manager and Output Module focuses on saving the system's data and creating logs. The File Input Module manages the files the user passes through to begin the "cold calling" functionality of the system. This functionality has a concept that is unrelated to the other modules and is therefore separate.

## 4.4. Data Manager and Output Module

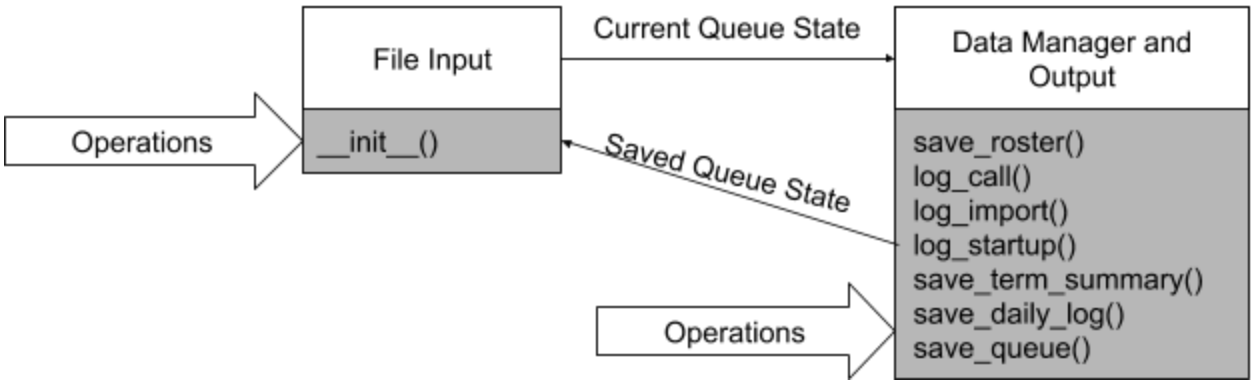
***The module's role and primary function.***

The Data Manager and Output Module's primary purpose is to save the system's data to allow the user to resume a previous saved state of the program. This module contains the formatting and saving of the data generated by the system.

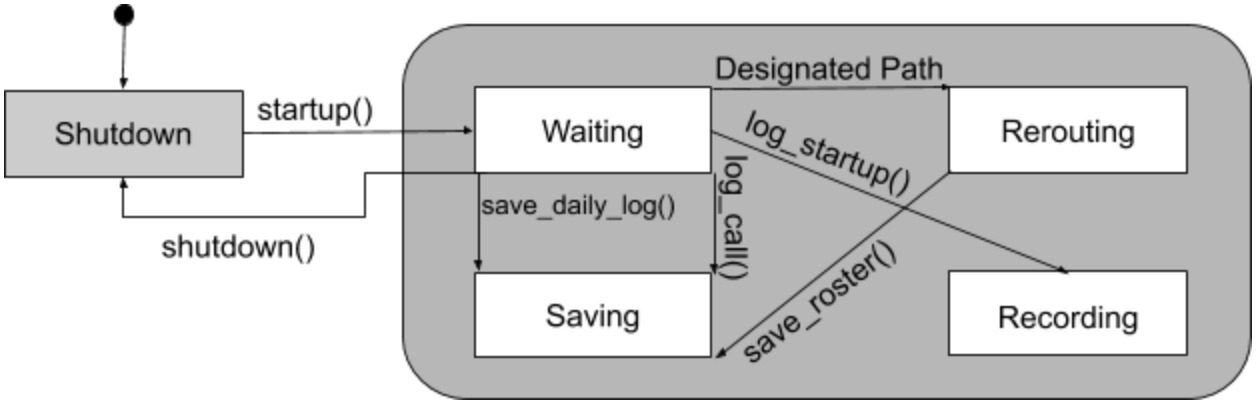
***Interface Specification***

This module interacts with the following modules: the File Input Module. The File Input Module sends the current queue state to the Data Manager and Output Module and the File Input Module receives the saved queue state from the Data Manager and Output Module.

***A Static and Dynamic Model***




(Figure 4.4.1) Data Manager and Output Module UML Class Static Diagram

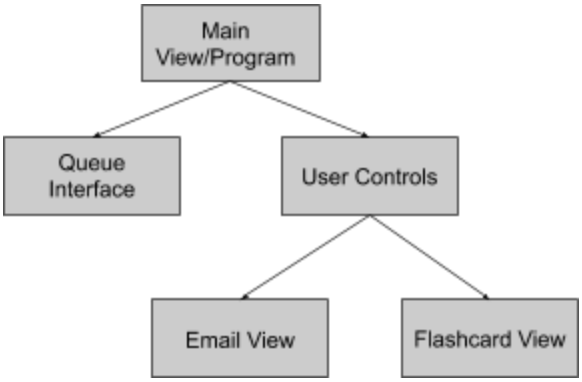



(Figure 4.4.2) Data Manager and Output Module UML State Dynamic Diagram

***Design Rationale***

This module is designed to control saving the system's data and manage the program's output. It saves the state of the queue, the order of the class list, and creates the session and term logs/documentation. This functionality was made into its own module to keep the saved state of the system's data and the output logs separate from the rest of the program. This leads to less errors when formatting the output files and restarting the program based off of the saved data. This module also has weak cohesion and is loosely coupled with the other modules. The User Interface Module focuses on the graphics and controls of the system, the Queue Algorithm and Manipulation Module focuses on the queue data structure, and the File Input Module focuses on parsing and formatting input files. The Data Manager and Output Module manages the output logs and the saved state of the "cold calling" program. This functionality has a concept that is unrelated to the other modules and is therefore separate.

## 4.5. Alternative Designs



(Figure 4.7.1) Alternative/Initial design of subsystems.

Figure 4.7.1 displays our initial design for a "cold call" system. Our group did not implement this design due to various structural disadvantages that did not match our group's development style. Foremost, our group decided to dismiss the production of the Email View and Flashcard View. The Email View allows the user to email the students that were flagged during the previous "cold calling" session. The Flashcard View allows the user to learn the names of the students in their class through a flashcard program designed much like Quizlet. This architecture isolates both of these views to account for the case that the time needed to implement them is not alloted. In our current architecture, these two modules have been removed due to these time restraints. Also, the above architecture separates the user interface and the user controls. The Python Tkinter library allows the interdependence of the user interface with the user controls. Due to the previous architecture disadvantages, our group decided to go with the architectural design modeled in Figure 3.1.

# 5. Dynamic Models of Operational Scenarios (Use Cases)

There is one use case for this system. This use case is the "cold calling" functionality of creating a queue of students that are "on deck" to be called on in a class setting.

The "cold call" use case goes as follows:

1. The user opens the application by clicking on the *ColdCall* file and the user interface is displayed. This action runs the *ui.py* file and waits for a file to imported to use for the queue
2. The user selects the Import button and selects a file to use as their class's roster. This action gives the *create_queue.py* file an input file to parse, shuffle, and then generate a queue of students to be "cold called" on. It then sends this queue to the *ui.py* file to be displayed for the user.
3. The user selects a student by using the left and right arrows and removes a student by pressing the up arrow. This action sends the removed student to the *create_queue.py* file to be removed for the queue. The number of times the student has been "cold called" on is increased by one. The queue is then shifted over to create space for a new student to be

added to the queue. A new student is then added to the queue and the new queue is sent to the *ui.py* file to be displayed.

4. The user selects a student by using the left and right arrows and flags a student by pressing the up arrow. This action sends the flagged student to the *create_queue.py* file to be removed and flagged for the queue. The number of times the student has been flagged is increased by one and the number of times the student has been "cold called" on increases by one. The queue is then shifted over to create space for a new student to be added to the queue. A new student is then added to the queue and the new queue is sent to the *ui.py* file to be displayed.
5. The user closes the system. This action sends the shutdown call to the *ui.py* file which is then sent to all the other files within the system. The *save_data.py* file responds to this shutdown call by saving the state of the queue, and the order of the class list in a separate file to be used when the system is restarted.
6. The user can now restart the program by repeating step one. The queue will restart where it left off and the user can skip step two.

# 6. References


"10.4: Hide Terminal's Window during Launch at Login - Mac OS X Hints." *RSS*, http://hints.macworld.com/article.php?story=20050512095145996

Bryan Oakley. "How to Get the Tkinter Label Text?" *Stack Overflow*, 1 May 1961, stackoverflow.com/questions/6112482/how-to-get-the-tkinter-label-text.

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.

Fiver. "Tkinter - Can't Bind Arrow Key Events." *Stack Overflow*, 1 Nov. 1963, stackoverflow.com/questions/19895877/tkinter-cant-bind-arrow-key-events.

freeCodeCamp.org. "Tkinter Course - Create Graphic User Interfaces in Python Tutorial." *YouTube*, YouTube, www.youtube.com/watch?v=YXPyB4XeYLA.

Hans van Vliet. (2008). *Software Engineering: Principles and Practic*e, 3rd edition, John Wiley & Sons.

IEEE Std 1016-2009. (2009). IEEE Standard for Information Technology—Systems Design—Software Design Descriptions. https://ieeexplore.ieee.org/document/5167255

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12), 1053-1058.


# 7. Acknowledgments

This template was modeled after Anthony Hornof's base Software Requirement Specification (SRS) given to the UO class CIS 422 in Winter 2020. As acknowledged in Hornof's acknowledgements section:
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.

The project creation process is based off of ideas found in the textbook: Hans van Vliet. (2008). *Software Engineering: Principles and Practic*e, 3rd edition, John Wiley & Sons.