CIS 422 Project 1:
Ducks on Deck
Software Design Specification

Kalyn Koyanagi (kek), Ellie Kobak (erk), Kelly Schombert (ks), Liza Richards (ljr) - 1-30-2022 - v2.0

Table of Contents

1. SDS Revision History

Date	Author	Description
4-30-2019	ajh	Created the initial document template.
1-6-2022	kek/erk	Removed content in Dynamic Models
1-9-2022	erk	Added UML Diagram to section 5.1
1-9-2022	kek	System Overview section added
1-10-2022	kek	Software Architecture section added
1-10-2022	erk	Section 4 added
1-11-2022	erk/kek	Finished first working draft
1-24-2022	ljr	Made changes to section 2, 3, and 5
1-29-2022	ljr	Made changes to Sections 3 and 4.
1-30-2022	ks	Redid UML diagrams.

2. System Overview

This software provides instructors the ability to “cold call” students in a live or virtual setting. Running the cold-calling system leads to the following sequence of events:

1. Upon starting the program, the user is prompted with a home screen that includes the ability to export class roster data, start a new class session, or exit the system.
2. When a new class session is started, the software asks the user to import a class roster text file. Once a valid file (meaning the file is of the right type and format) is submitted, the system will put the student names in a queue and randomize them.
3. Four names are chosen randomly from the queue and are displayed in the on deck window for the user to select from to call on.
4. The software tracks which students have been called on and which students have been flagged and provides this information in the form of text logs for later viewing by the instructor.

The system is organized into five different components that are all responsible for interacting with each other to interpret the user data for an “on deck” session. Each component handles one of the following:

1. Reading data
2. Building a queue
3. Data writing
4. Visuals and user interface

The components that deal with reading data and building a queue work very closely as the queue is built using the imported data. The data writing component is called by each of the other components. The queue component calls the data writing component when a name is called on and removed from the “on deck” list. This also updates the visual component. All of the components are linked together by the visual component because of the user input.

3. Software Architecture

The functionality of the “cold calling” system is broken up into five different components/files.

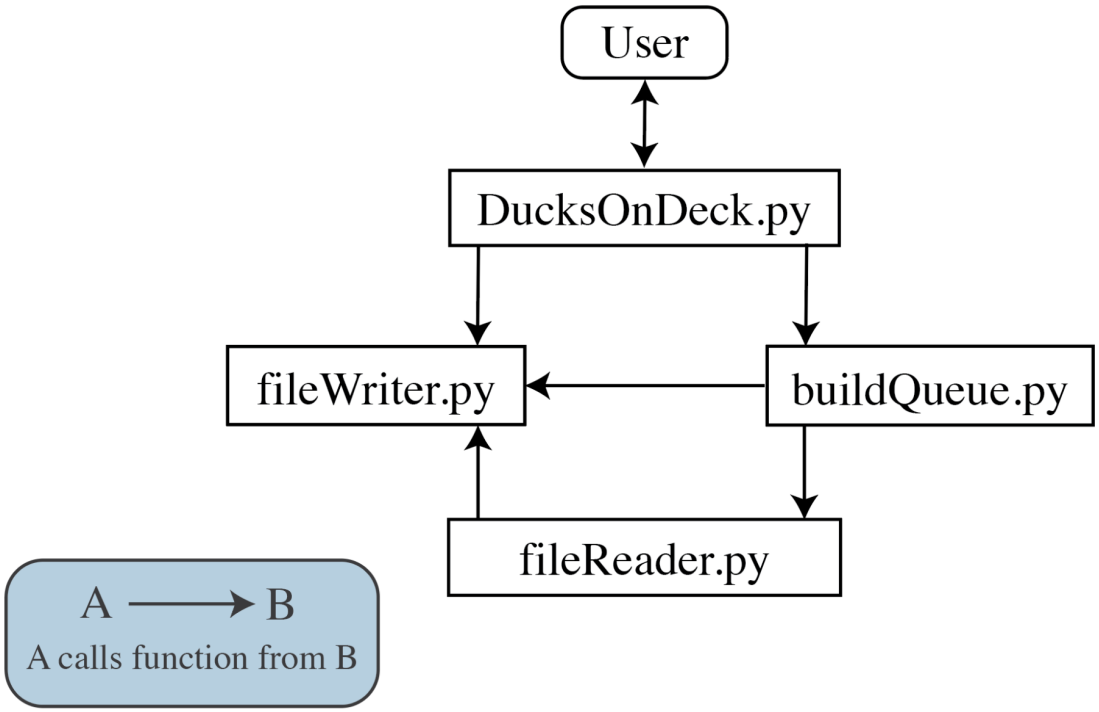

Figure 1. Components of Ducks on Deck

3.1. Components:

1. buildQueue.py: uses the imported class data to place the students into a randomized queue, which will be used for selecting students to be “on deck.”
2. fileReader.py: opens and reads the text files imported by the user and stores all relevant data.
3. fileWriter.py: writes the daily class data in a text file that can be exported and downloaded by the user.
4. DucksOnDeck.py: responsible for both the display windows and the key controls used by the instructor.

3.2. Component Interactions:

The DucksOnDeck.py is responsible for all the functionality of the software and acts as a driver for running the program. Once the program begins, buildQueue.py is immediately called. Then, buildQueue.py calls the fileReader.py in order to obtain the file with the class roster. Once a valid file has been entered, the Ducks On Deck main menu appears for the user to interact and use for cold calling.

The DucksOnDeck.py handles the keystroke and user mouse input and corresponds what has been pressed to either the buildQueue.py or the FileWriter.py. In order to start cold calling, the user must press ‘New Session’. This creates another pop up which calls the onDeck function from buildQueue in order to display the top four names in the

queue. Every removal of a name using the keystrokes signals the buildQueue.py and the FileWriter.py. The buildQueue.py keeps track of which names need to be removed and the queue data structure. The FileWriter.py uses a dictionary to record who has been called upon and flagged to access the data later. The buildQueue.py also signals to the FileWriter.py when a name has been removed from the Deck and updates the information logs. The other two components the user can click on are exit, which terminates the program, and 'Export Performance Summary' (EPS). The EPS connects the Ducks On Deck to the FileWriter.py. The FileWriter.py also takes user input to name new files that store the term and daily summaries from the cold calling data.

3.3. Design Rationale:

1. buildQueue.py: The queue data structure is critical for holding and organizing each student's information that is used to choose who will be on deck. It is the most efficient and simplest way to ensure organized yet random selection of the students.
2. fileReader.py: Reading from the files is essential for obtaining each student's information and storing it in a way so that it may be used by the system. It is its own separate component from writing a file so that the original files are not changed.
3. fileWriter.py: The system requires for there to be a way to record all the cold-calling data that will be collected during run time. This file is responsible for making the appropriate changes to the files as the data is collected. It is its own separate component so that the original class roster file is left unchanged and does not interfere with this one.
4. DucksOnDeck.py: The DucksOnDeck file is responsible for all of the visual components of the system. The system requires for there to be a window displayed on the user's screen with the option to start a new cold-calling session, export a summary performance, or exit the system. This GUI component will allow the user to choose between students to call on using input commands which can be recorded.

4. Software Modules

4.1. Visual Interface

The module's role and primary function.

The purpose of the Visual Interface is to allow user interaction from the instructor with the program. One of the most important aspects of this interface is the GUI visual component, which displays the user input on a window. The user input is then processed through both the deck, flag logging, and the export of data as requested by the instructor. This component has the following responsibilities:

1. Displays a list of four students in the On Deck window to be chosen from to cold-call on
2. Uses keyboard input from the user to flag & remove or only remove a student from the On Deck window

Interface Specification

The GUI enables the user to see and interact with the two components of the program. The first is the deck, which displays the top four names of the class roster queue. The second is so the instructor is able to interact with the file

exports. The file export is how the instructor can view the cold calling data collected on any day or throughout the term as described in the use cases of the SRS and in section 5.1.

1. DucksOnDeck.py calls upon buildQueue.py to create a Queue using student class data.
2. fileWriter.py is used by DucksOnDeck.py to export the performance summary log when prompted by the user.

Models

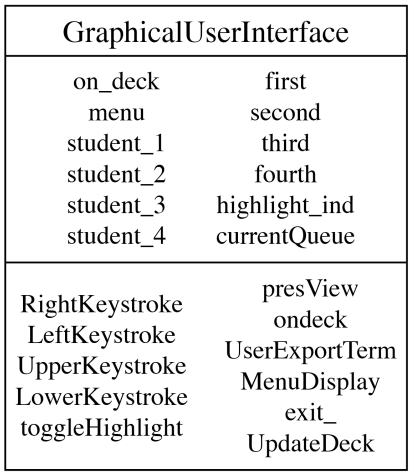

Figure 4.1.1 Static Model of Visual Interface

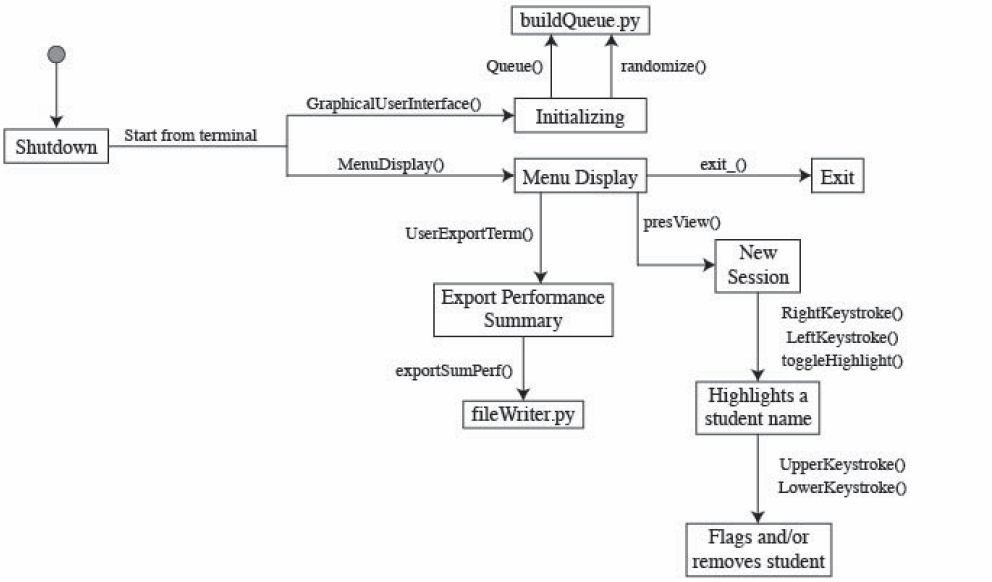

Figure 4.1.2 Dynamic Model of Visual Interface

Design rationale

The intention of designing the Visual Interface is to demonstrate how the instructor or user is able to interact with the program. Thus, a central component of the design is for the GUI and visual aspect to be connected to all other functionality described in the use cases from section 5 and in the SRS.

4.2. Queue Interface

The module's role and primary function:

The primary functions of the Queue Interface are:

1. Create a queue from the list of names given by the imported file.
2. The queue is then sent to the reader, where the names are displayed on the deck.
3. Within the Queue, the names from the class roster are randomly sorted.
4. The Queue removes and then adds names to the end of the Queue as directed by the keystrokes.
5. Sends information of who has been removed from deck to the File Writer.

Interface Specification

The Queue is the central component of the Ducks on Deck program. The Queue is structured as a class and the methods of the queue class get called upon in several other modules.

3. Queue calls FileReader to obtain data to create the queue.
4. Queue remove method is called by DucksOnDeck.py to remove names from the deck.
5. DucksOnDeck.py calls 'On Deck' Queue method, on Deck, to get names displayed on the visual on deck list.

Models

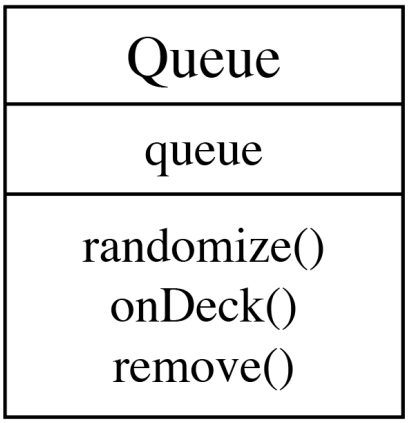

Figure 4.2.1 Static Model of Queue Interface

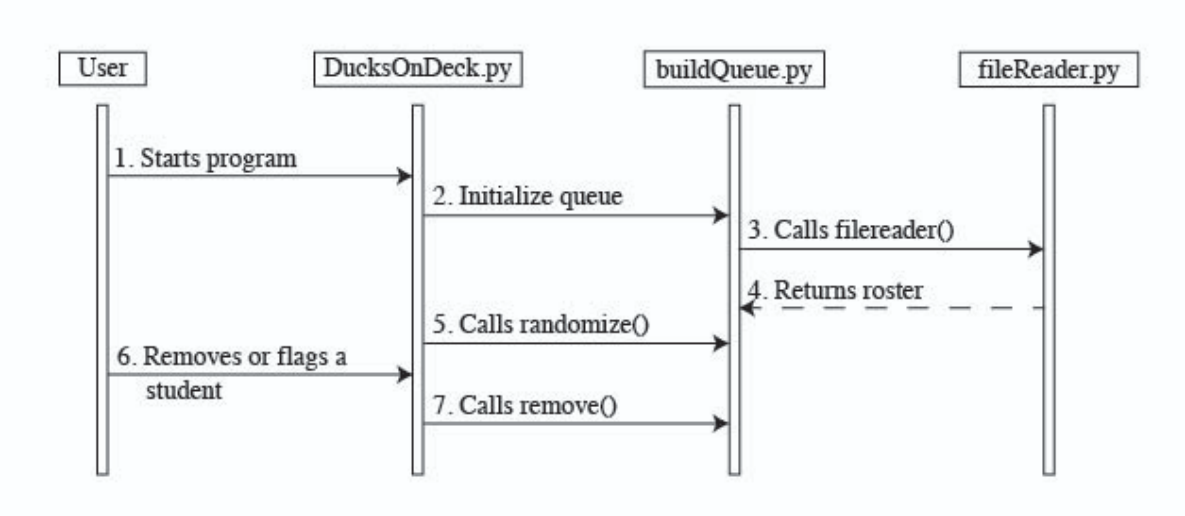

Figure 4.2.2 First Dynamic Model of Queue Interface

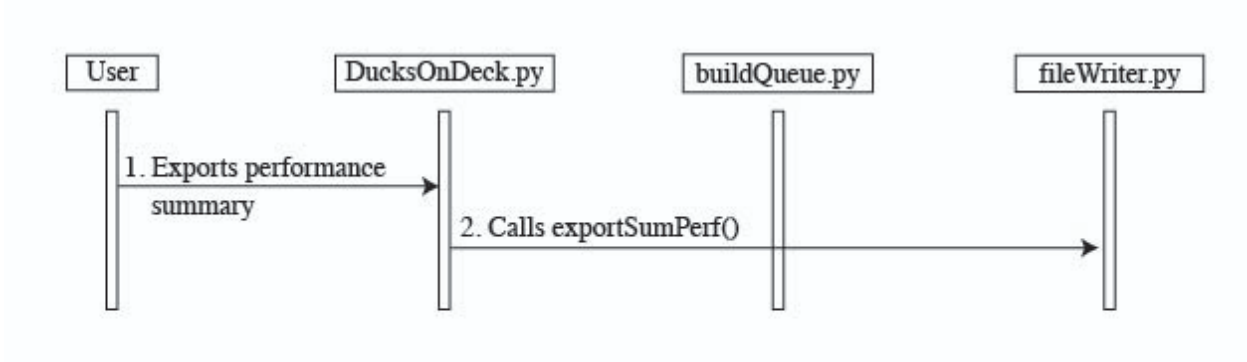

Figure 4.2.3 Second Dynamic Model of Queue Interface

Design rationale

When the student list file is imported into the software, the system will in turn parse its contents into a queue. From there, students will be randomly chosen to be in the on deck list from the front n% of the queue. Students are only chosen from the front n% of the queue because otherwise once students are rotated in the queue, the back portion will be students that were too recently chosen to be on deck. This method ensures the equal distribution of the number of times students are called on. From the queue, 4 students are chosen to be on display in the on deck list for the reader to see and choose from.

4.3. File Reader System

The module's role and primary function.

The file usage encompasses fileWriter and fileReader which is how the imported and exported files are used in the system. It explains how the importable and exportable files are connected to the rest of the program. The following list explains the different responsibilities of the fileWriter and fileReader:

1. The file reader takes the tab-delimited file of the class roster and sends the parsed information to the queue.
2. The queue then randomizes the order and sends the names to the user interface/file writer, which allows the data to be collected from participation and use.
3. The data is then sent to the file writer, where all of the information on who is flagged, who has spoken, and how many times one has spoken is stored.
4. This data is then accessible through the logs, which are automatically updated from the file writer.

Interface Specification

The File Reader's main function is to parse the files inputted by the user into usable, mutable lists. This is done through the following:

1. When the queue is initialized, the queue calls the File Reader to get the class roster.
2. The File Reader sends the unchanged order of class roster to File Writer to initialize performance summary

Models

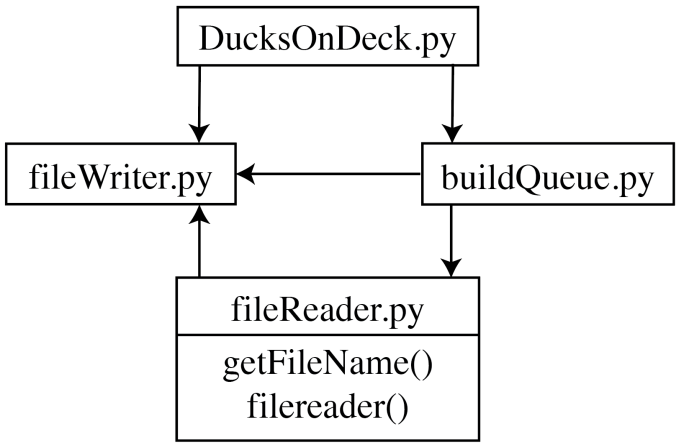

Figure 4.3.1 Static File Reader Model

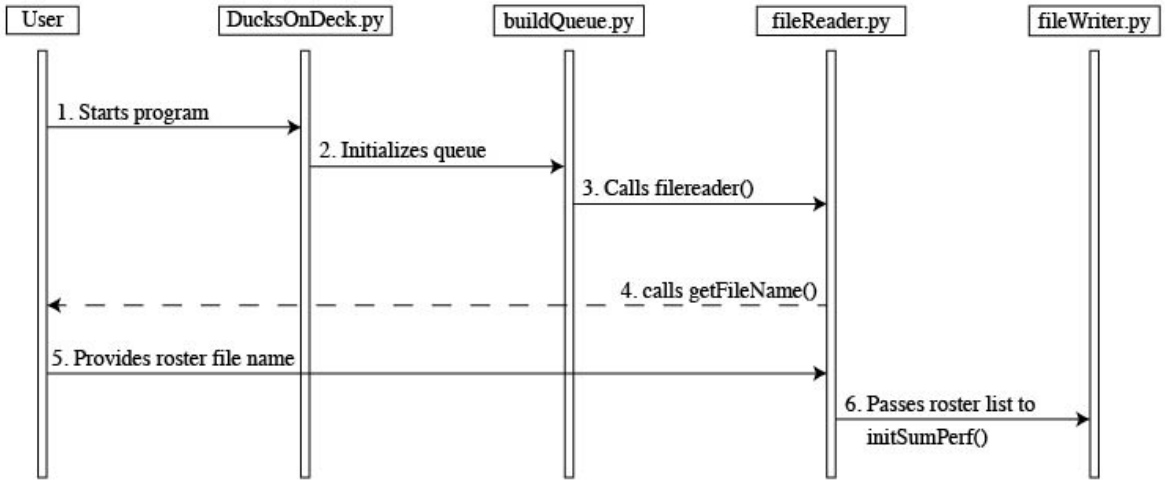

Figure 4.3.2 Dynamic File Reader Model

Design rationale

The static file usage model (shown in figure 4.3.1) demonstrates how and which files are exported and inputted. Main will read the file into a queue in which the user will be able to write to the files as the system is used. At any point when the program is running, three main files are available for the user to access. First you will see the flag log, which will show the user which students they chose to flag and why. They will also see a daily log file where the user will be able to see all the cold-calling data collected for that day. Lastly, the user will have access to a term log file which is a summary of all the cold-calling data collected for each student from the entire term. The dynamic file usage model shows how the user will be able to continuously modify those three files throughout runtime using the specified keystrokes by importing and exporting them repeatedly.

4.4. File Writer Interface

The module's role and primary function.

The primary functions of the File Writer Interface are listed below:

1. Keep track of all of the information from the instructor controls in a stored manner that can be exported and easily viewed.
2. Demonstrates how the queue interacts with the record, so all data on cold calling is properly saved.

Interface Specification

The File Writer exports the data of who was called on during a session and if they were flagged. The File Writer constantly interacting with other components of the program in the following ways:

1. The File Reader calls the File Writer to initialize the file that will contain the cold calling data.
2. The Queue calls the File Writer every time a name has been removed from the deck. This ensures all data is tracked and stored at all times.

3. DucksOnDeck.py calls File Writer to export the summary performance files.

Models

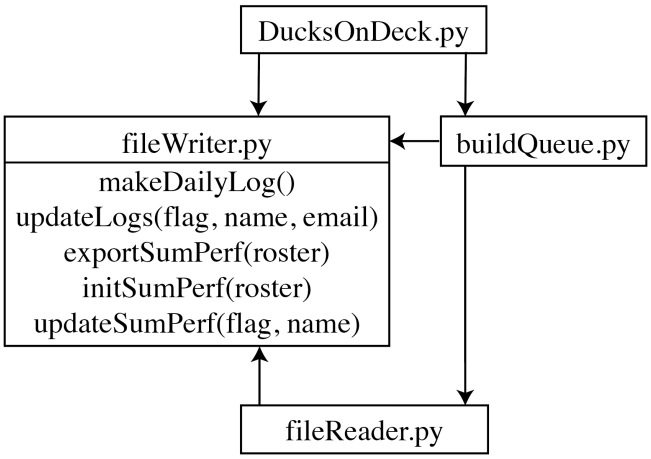

Figure 4.4.1 Static Model of File Writer Interface

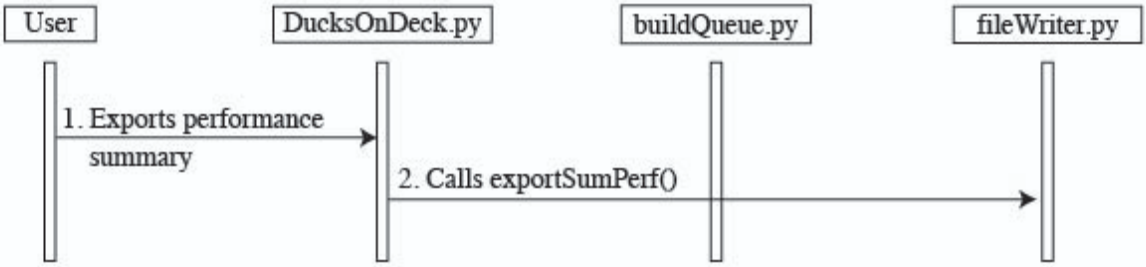

Figure 4.4.2 Dynamic Model of File Writer Interface

Design rationale

Once files are imported into the system, they are parsed into a queue. In addition, there is a record kept for the term log, daily log, and flag log files. These are files that contain past cold-calling information to be reviewed by the instructor. Once the contents of the files are in queues, the user will operate the arrow keys on their keyboard to move around the deck which contains four random students chosen from the queue. While being on deck, students can be removed or flagged. If removed, the student will be enqueued onto the back of the queue. This action results in data being added to the term log and the daily log, but not the flag log. If the student is flagged, then they will not only be removed from the deck, but their flag data will be added to the flag log file now in addition to the term log and the daily log files.

5. Alternative Design

An alternative design that was considered for this system was to have all of the existing components of the system be integrated together in a main module. The main module would be the driver for running the program. It would have been responsible for the functionality of the software and would interact with all of the modules in the architecture. The interactions between the main module and the other existing components is listed below:

1. The main module would first call fileReader.py to parse and import the data to the “On Deck” window list.
2. The visual component (DucksOnDeck.py) would be called next to display the “On Deck” window and allow user interaction using keyboard input.
3. For the user to export the collected cold-call data, DucksOnDeck.py would indicate a final update to the main module
4. The main module would then call on the fileWriter to write the final update to a daily and term log to be exported

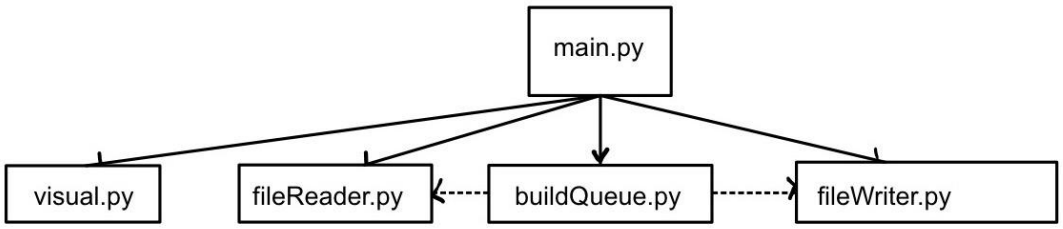

Figure 5.1.1 Unused Model Representing Main Module Interaction

We found that with the current system, we could run the program through the visual component (DucksOnDeck.py) without the use of a main module. DucksOnDeck.py incorporates all of the other modules successfully by itself. As a result, we decided to not incorporate the main module at all.

5. Dynamic Models of Operational Scenarios (Use Cases)

There is one defined major use case for this program. This primary use case consists of the cold calling “on deck” system, which most users utilize.

5.1. Standard “Cold Calling” Use Case

The standard instructor use case is as follows:

1. Instructor runs cold-call-assist software from the terminal on the computer
2. Cold-calling main menu is displayed on screen
3. User selects “New Session”
4. System loads a provided class list into a queue data structure

5. System displays a new “On Deck” window with four student names displayed
6. The instructor can move between any displayed student to call on using left and right arrow keys
7. Instructor chooses a student and either removes them or flags them
8. Number of cold-calls and flag information written and saved to output files
9. Chosen student name will be removed and placed back into the queue
10. System will replace old name with a new and random student name from the queue
11. The system will re-jumble the names between runs
12. Instructor exits system
13. Output file is saved
14. Instructor reviews daily log file
15. Instructor reaches out to flagged students
16. System is restarted and is ready for the next run.

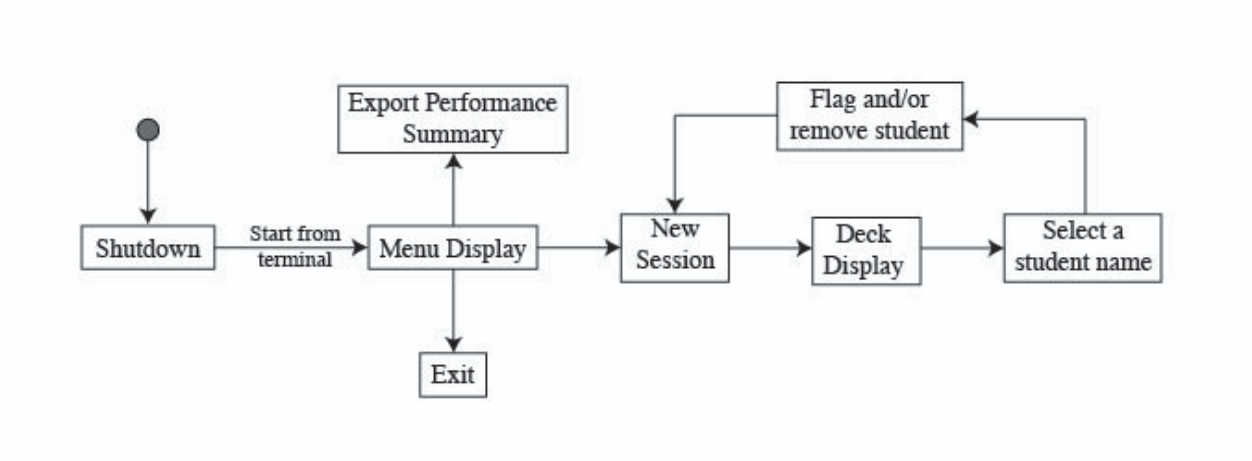

Figure 5.1 UML of System Usage

5.2 Flagged Students Use Case

The flagging system operates as follows:

1. Instructor runs cold calling system
2. Instructor flags students in the On Deck list
3. Instructor ends system runtime.
4. Instructor reviews flagged students in the daily log file
5. Instructor reviews what notes were made on the flagged students
6. Instructor uses student emails in the class list to email students on topics of corresponding interest.
7. Instructor finishes emailing students and exits system entirely

5.3 Instructor Reviews and Shares End of Term Summary

The instructor follows these steps to obtain and share end of term cold-calling summary data:

1. Instructor opens system
2. Instructor chooses “Summary” button from main menu

3. End of term summary file is displayed with complete number of cold calls and flags per student
4. Instructor saves the summary file to their computer
5. Instructor shares file to class through email
6. User exits system entirely

6. References

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. Acknowledgements

This template was given to the UO CIS 422 class by Anthony Hornof. This template is similar to a document produced by Stuart Faulk in 2017, and uses publications cited within the document, such as IEEE Std 1362-1998 and ISO/IEC/IEEE Intl Std 29148:2018.