CIS 422 Project 1:
Ducks on Deck
Software Design Specification

Kalyn Koyanagi (kek), Ellie Kobak (erk), Kelly Schombert (ks), Liza Richards (ljr), Luying Cai (lc) - 1-6-2022 - v1.0

Table of Contents

1. SDS Revision History 1

2. System Overview 1

3. Software Architecture 2

4. Software Modules 3

4.1. Instructor Interface 3

4.2. Queue Interface 3

4.3. File Usage 4

4.4. Record System 5

4.5 Alternative Designs 6

5. Dynamic Models of Operational Scenarios (Use Cases) 7

5.1 Standard “Cold Calling” Use Case 7

6. References 8

7. Acknowledgements 8

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

2. System Overview

This software provides instructors the ability to “cold call” students in a live or virtual setting. Upon starting the program, the user is prompted with a home screen that includes the ability to import and export class roster data and the ability to start a new class session. When a new class session is started, the software randomizes the student names off of a roster (either previously saved or newly imported) and inserts them into a queue. The first five student names in the queue will be displayed as the “on deck” students who may get called on by the instructor. The software tracks which students have been called on from the “deck” and which students have been flagged and provides this information in the form of 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: reading data, building a queue, data writing, visuals and user interface, and passing information between files. The components that deal with reading data and building a queue work very closely as the queue is built off of the imported data. The component concerned with the visual and user interface works closely with the writing data component as any change made by the user will immediately get written to a log. These two groups of closely working components are linked by the final component, which will ensure that data from the queue is sent to the visual component for proper display.

3. Software Architecture

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

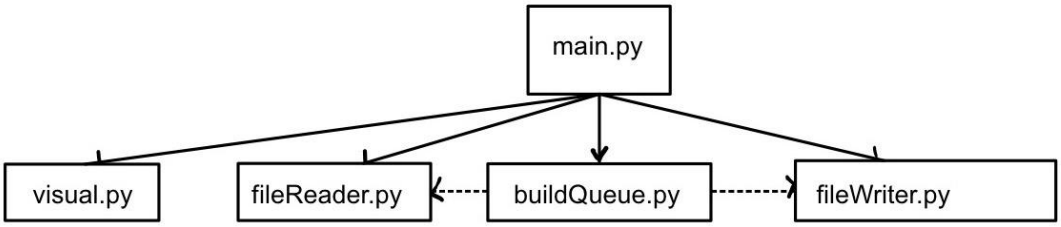

Figure 1. Components of Ducks on Deck

Components:

1. main.py: brings all of the functionality of the other files together and is the driver for running the program.
2. buildQueue.py: uses the class data from fileReader.py and places the students into a randomized queue, which will be used for selecting students to be “on deck.”
3. fileReader.py: opens and reads the text files imported by the user and stores all relevant data.
4. fileWriter.py: writes the daily class data in a text file that can be exported and downloaded by the user.
5. visual.py: responsible for both the display windows and the key controls used by the instructor.

The main.py component is responsible for the functionality of the software and interacts with all of the modules in the architecture. As a file is imported by the user, main.py calls upon fileReader.py to parse and import the data for an “on deck” session. Once the student info is read, buildQueue.py uses the data to efficiently place the students in a randomized queue for the “on deck” selections. To control the user interface, visual.py interacts with main.py to display the main user window as well as the “on deck” display. The visual.py file also controls how keyboard clicks are interpreted by the system. When the user closes the program, visual.py indicates a final update to main.py, which then calls upon fileWriter.py to write the final update to a daily and term log and export the user data. The fileWriter.py component is also responsible for making sure user data is saved after every user interaction to ensure no vital information is lost in the case of a fatal system crash.

Design Rationale:

The architectural design of this system is centered around simplicity. The rationale behind the choice of these components is to encapsulate all of the primary functions needed for the software while eliminating any potential redundant code. The compartmentalized files allow for easy testing and debugging, as every module can be tested individually. The current software architecture minimizes the need for transferring user data between components, which ensures user data security and efficiency. Maintaining efficiency is a major element of the architecture, as speed will often be a top priority for instructors using Ducks on Deck in a live classroom setting.

4. Software Modules

4.1. Instructor Interface

The module's role and primary function.

The purpose of the Instructor 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.

Interface Specification

The GUI enables the user to see and interact with the three components of the program. The first is the deck, which displays the top four names of the class roster queue. The second is flag exports. This is for the instructor to flag a student's name for any reason highlighted in the SRS. This is possible because the GUI enables users to use keyboard input to create flagging data.. Third, 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.

Models

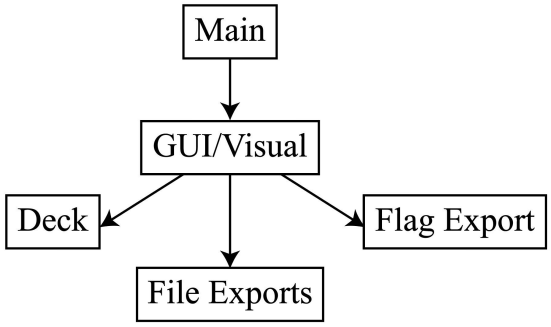

Figure 4.1.1 Static Model of Instructor Interface

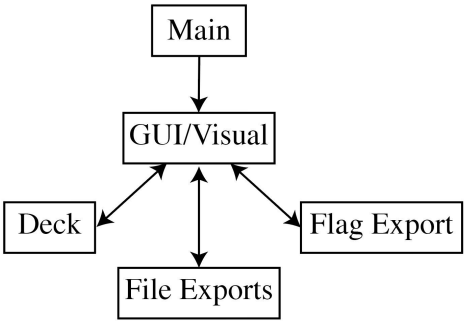

Figure 4.1.2 Dynamic Model of Instructor Interface

Design rationale

The intention of designing the Instructor 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 function of the Queue Interface is to create a queue from the list of names imputed. The queue is then sent to the reader, where the names are displayed on the deck. Within the Queue, the names from the class roster are randomly sorted.

Interface Specification

In the main program, both the Queue and the Reader are called and run. The Queue only stores the class roster in a randomly generated order. The list of the roster is then sent to the reader. In the reader, the top four names on the Queue are displayed on the deck for cold calling. After the names are removed from the deck, the information is sent back to the queue for future cold calling.

Models

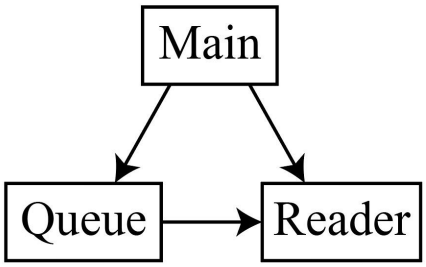

Figure 4.2.1 Static Model of Queue Interface

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 Usage

The module's role and primary function.

The File Usage explains how the exportable files are connected to the rest of the program. The File reader takes the tab-delimited file of the class roster and sends the parsed information to the queue. 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. 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. This data is then accessible through the logs, which are automatically updated from the file writer. The three logs can be accessed at any point by the user.

Interface Specification

In the File Usage, the first interaction is between the main interface and the file reader. Main is the driver for the function and when it prompts the user to input a file. Once the file is inputted, it can be read and parsed through being sent to the queue. After the names of students are randomly sorted, the queue sends these names to the file writer. In the File Writer, each log has a file created and or updated automatically while the user is interacting with cold calling students.

Models

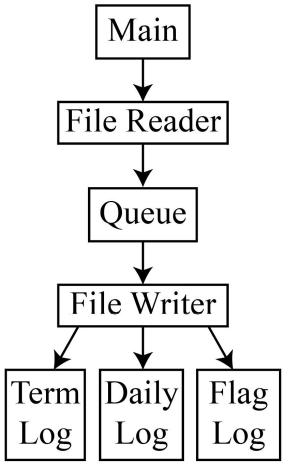

Figure 4.3.1 Static File Usage Model

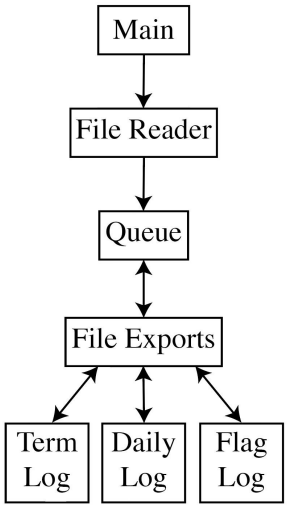

Figure 4.3.2 Dynamic File Usage Model

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. Record System Interface

The module's role and primary function.

The primary function of the Record System Interface is to keep track of all of the information from the instructor controls in a stored manner that can be exported and easily viewed. Thus the module demonstrates how the queue interacts with the record, so all data on cold calling is properly saved.

Interface Specification

The data is recorded from the main interface running the Queue and Record simultaneously. Within the Queue, there is the list, which contains the entire class roster, the deck, for the four names to be called upon, and the controls for the user. The deck then allows access to the flag or removes a name from being cold called. These two actions both signal the associative logs in the Record that an action has been done for the student called on. Therefore, the logs are constantly being signaled with updates when the cold call program is running. This ensures all data is tracked and stored at all times.

Models

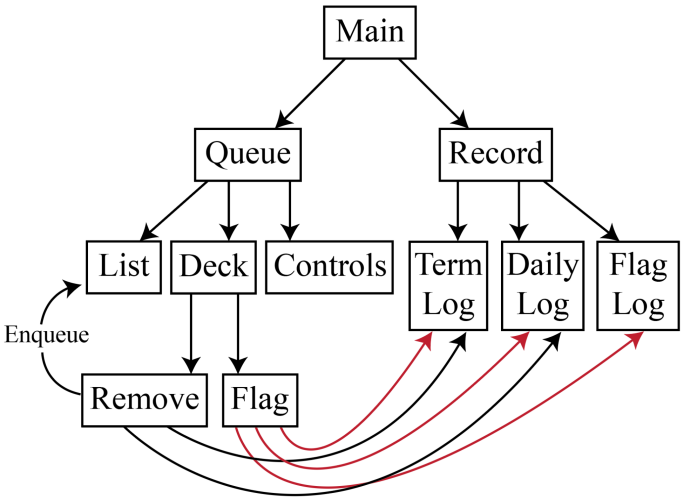

Figure 4.4.1 Dynamic Model of Record System Interface

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.

4.5. Alternative Designs

An alternative design to this interface would be to have the user be able to manually choose how many students they want to have on deck at one time. The base number would be 4, however if the instructor has a smaller or larger class size, they may want to change how many students they can have on deck at once. To incorporate this, the user would be asked to manually enter the number of on deck positions they want right after startup. From there, the amount of on deck positions on display would correspond with the number entered by the user.

Figure 4.5.1 Alternative Model of Program Design

We did not use the model from figure 4.5.1 because there were unnecessary dependencies between a name being flagged and being removed.

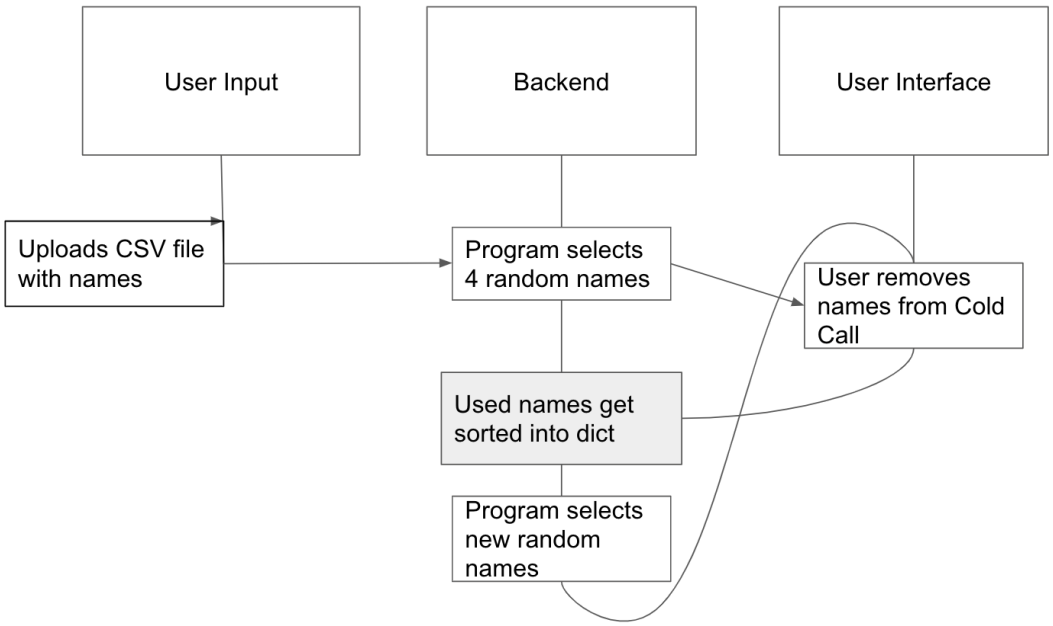

Figure 5.1.2 Unused Model Representing Recording and User Interaction

The primary reason we did not pursue the idea behind the model in figure 5.1.2 is due to not using a dictionary to keep track of if a name is flagged or called. Although we initially thought the loop between the names sorted to be sent to the user interface, we quickly realized this model hugely oversimplified how to execute this idea.

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 two ways an instructor can use the program is the following:

1. Daily instructors use cold-call-assist software installed on their laptop. After the instructor opens the software, it will automatically be positioned on the top of PowerPoint with four names displayed. The instructor can choose any displayed student to call on, and if they respond then the instructor has the option to flag and remove a student or just remove them from the displayed list. If the instructor flags a student, they can write a note regarding their response. To raise a “flag” for a student, the instructor can use arrow key input. The system will re-jumble the names between runs while making sure students who were not cold called at the previous class will be prioritized. After class, the instructor can easily copy students’ information to write an email to contact the flagged students.
2. After checking the cold call log file, the instructor can send emails to the flagged students with notes in the response code, otherwise the response code will have “X.” The instructor can use the copy command to copy a student’s email address.

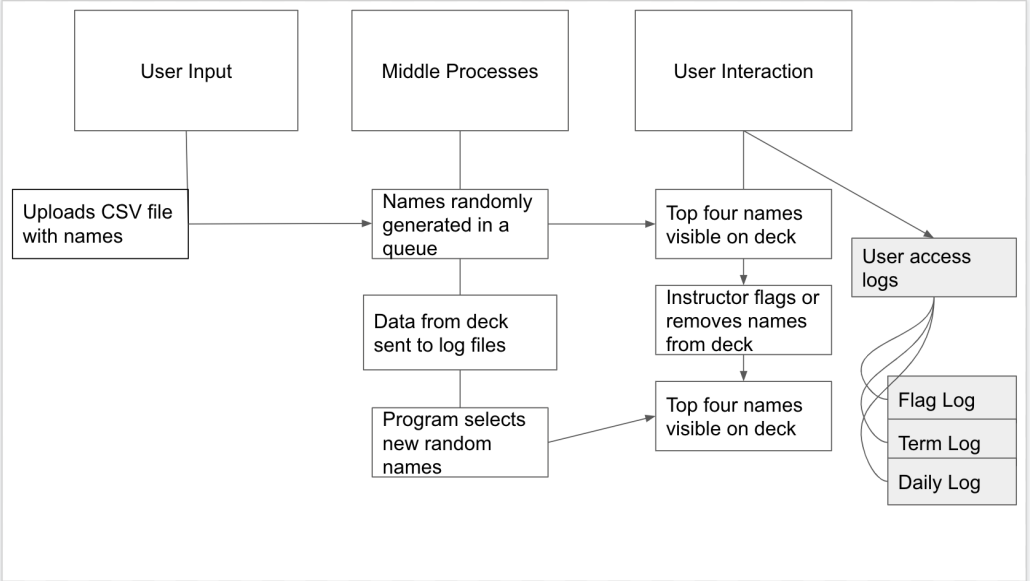

Figure 5.1 UML of System Usage

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

The template of this document was based on a Software Development Specification template provided by Professor Anthony Hornof.