Cold Calling

Software Design Specification

Table of Contents

1. SDS Revision History

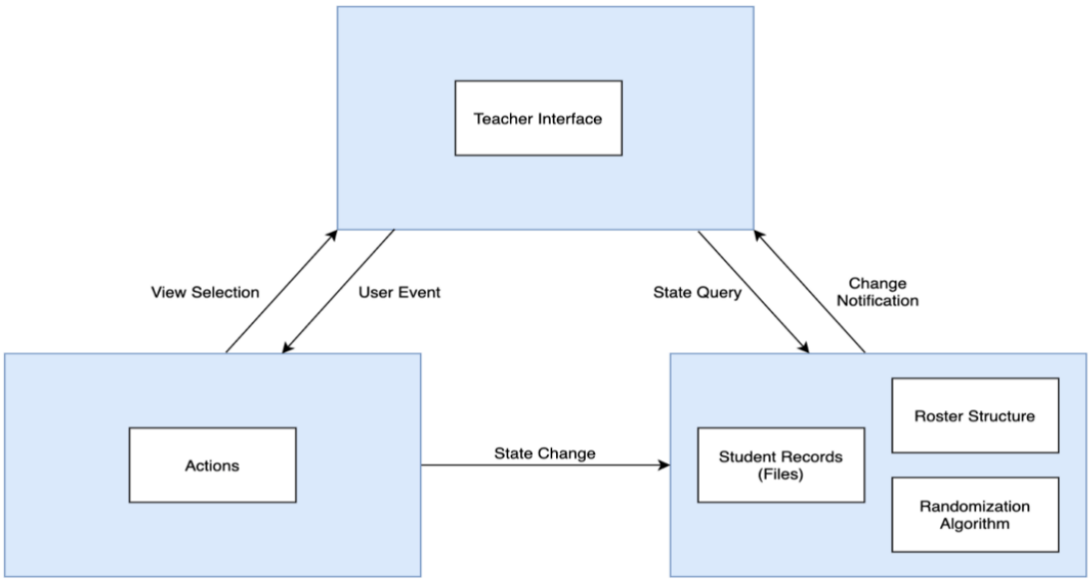
Date	Author	Description
1-14-20	MM	Created the initial document.
1-15-20	LH	Section 5 diagrams.
1-15-20	NT	Section 3 diagrams and descriptions.
1-16-20	YJ	Section 4 module descriptions.
1-16-20	MM	Edited, rewrote and reformatted.
1-16-20	All staff	Generate v1 of document.

2. System Overview

Cold calling is the method of asking a random group of students to provide input to a proposed question. The Cold Calling software system will read in a class roster as input and provide a randomized list of student names in an easy to view, compact, user interface. The services of this software include using arrow keystrokes to highlight, remove, and flag names. To highlight names on the "on deck" list, the user will use the left and right arrow keys. To remove a student's name after they have answered a question, the user will use the down-arrow key. Flagging student names signifies scheduling a follow up meeting. To flag a student, the user will use the up-arrow key; this key removes a student name as well as flagging a student for later discussion. The software randomizes the student information using a randomization algorithm, avoiding repetition across multiple days. A Summary Performance file will also be provided to allow the user to view which students have participated, which students are flagged, and which students have yet to participate. Along with a Summary Performance file, a Daily log file will be written every day to allow the user to view which students were flagged for that specific day. The Cold Calling software allows the user to generate student names, remove students that have spoken, flag students to follow up with, and view a feedback log of previous participation.

3. Software Architecture

Software Architecture



The components included in the Cold Calling software system are the teacher interface, actions, student records, roster structure and randomization algorithm.

- Teacher Interface - This is the graphical element that is visible to the user. The Teacher Interface lists the names of the four students that are “on deck”.
- Actions - This component handles user input, mapping the arrow keys to their corresponding functions.
- Student Records - The component acts as the database of the system. A roster file of student information is written in, and the data is written back out to a file following a user event (keystroke).
- Roster Structure - This component represents the queue of students to be called. The queue is comprised of Student objects, which encapsulate data such as the student name, uoID, etc.
- Randomization Algorithm - This component is the algorithm for reinserting a student into the queue.

These components interact with each other in many ways making the software robust, reliable and usable.

- The Teacher Interface sends user events such as keystrokes to the Action module.
- The Actions module selects or modifies the view of the Teacher Interface based on which arrow key was received.
- The Actions module triggers a state change in the Roster Structure, Randomization Algorithm, and Student Records modules.
- The Roster Structure, Randomization Algorithm, and Student Records modules notify the Teacher Interface of their changed state and can modify the information that it displays.
- The Teacher Interface can query the state of the Roster Structure for what names to display.

This architectural design is based on the Model View Controller architecture. This design allows for a clear separation between the Teacher Interface (the views), the Actions (the controller) and the Model, which is made up of the Roster Structure, Randomization Algorithm, and Student Records modules. These three modules are grouped together because they access and modify the same set of data.

4. Software Modules

4.1 Teacher Interface

a. The module's role and primary function:

The primary function of the Teacher Interface is the graphical element that is visible to the user. The Teacher Interface lists the names of the four students that are "on deck".

- Functionality:
  - The module will get a state Queue from the Cold Call Engine and receive a change notification
  - tkinter will also be used to develop the interface.

b. The interface to other modules:

The module will be able to provide an event (keystroke) to the action model and get a view selection back.

c. Static and Dynamic model:

The Teacher Interface should be a fully static model. We do not want any changes in the interface appearance.

d. Design Rationale:

The rationale for the design of the Teacher Interface is that the GUI must be implemented and tested without needing other modules.

e. Alternative design:

Communication directly from the Teacher Interface to the Cold Calling Engine.

4.2 Action

a. The module's role and primary function:

The Action module is playing the role of Controller in a Model View Controller architecture. This module will map the user actions in the Teacher Interface, to the Cold Call Engine. User actions contained in the Action module are up, down, right and left keystrokes. The Action module also includes importing and exporting files such as the Student Records and Summary Performance file.

- Functionality:
  - import(): import Student Records file.
  - export_daily_log(): Daily Log file.
  - export_term_log() Summary Performance file.

        - right_button(), left_button(), up_button(), down_button(): user action keystrokes.
        - startup(), shutdown(): start and exit the software.

b. The interface to other modules:

   The module will be notified if the user presses an arrow key in the Teacher Interface. According to the action, the Action module will tell the Cold Call Engine what to do.

c. Static and dynamic model:

   This module should be entirely static because every specific user action should map specifically to the Cold Call Engine.

d. Design rationale:

   Actions will be built after the Teacher Interface and backend modules are finished. The action module will be used to connect those two parts.

e. Alternative design:

   There are interfaces that work between backend (Cold Calling Engine) to GUI (Teacher Interface). An alternative design would be having the keystrokes directly influence the backend.

4.3 Cold Calling Engine

a. The modules primary function:

   This module contains three components; the Roster Structure, Student Records, and the Randomized Algorithm.

   - Roster Structure:
     - The Student Structure will be a self-built class that will contain student information. The structure will also contain helper functions such as:
       - printStudent(): prints student information.
       - display(): returns student name to be called by Teacher Interface.
       - summaryPerformance(): return student information to write to Summary Performance file.
       - review(): return student information to write to Daily Log file.
       - getflag(): was the student name flagged?
       - setflag(): flagging a student's name.
     - The Roster Structure Queue will be used to hold Student Records. The structure will also contain helper functions such as:
       - enqueue(): put a student name on to the queue.
       - dequeue(): pop a student's name from the queue.
       - insertOne(): inserts a new student at index i of the list.

        - removeIndex(): removes and returns a student from index i of the list.
        - isEmpty(): checks if the queue is empty.
        - combine(): combines a new list with the current queue.
        - printQueue(): prints the current queue.
    - Student Records:
        - The Student Records is the file I/O component. Using functions provided by the Roster Structure, the Student Records component will:
            - Read from a user import file or data stored file in tab-delimited format.
            - These records will be read into the Queue.
            - The Summary Performance file and the Daily Log file will be written to every time a student is removed or flagged.
    - Randomization Algorithm:
        - The Randomization Algorithm component contains the algorithm that will randomize the student names to be displayed on the "on deck" list. Using helper functions that the Roster Structure provided, the algorithm will produce four random names.

b. The interface to other modules:

   The Action module will notify this module of when and how to change state according to keystroke actions.

c. Static and Dynamic model:

   When you flag a student or try to see the state of the current roster this module should be static. The user is expected to see a specific thing. The dynamic part of this module is picking random names from the Queue.

d. Design rationale:

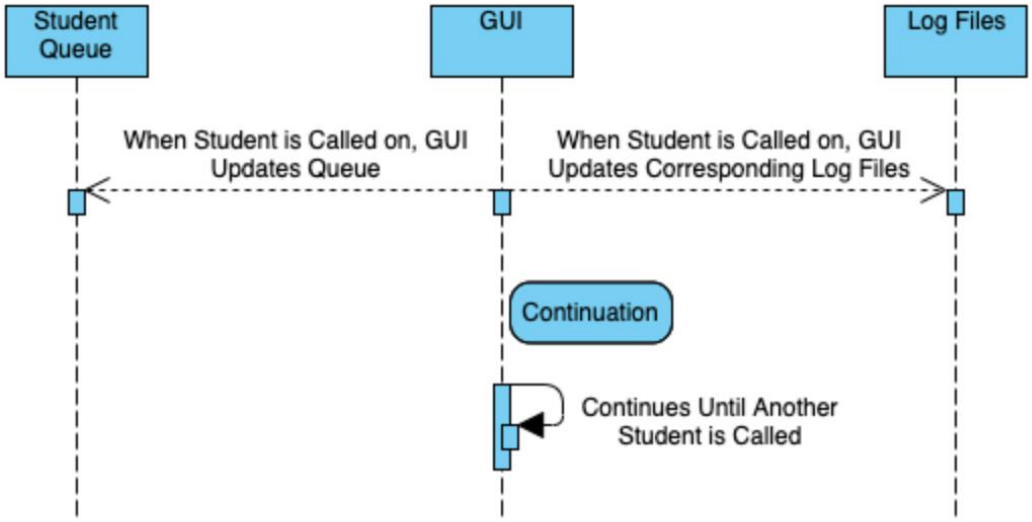
   These components are classified into one module because they represent the system backend. In other words, this module can be implemented and tested without the GUI.

e. Alternative design:

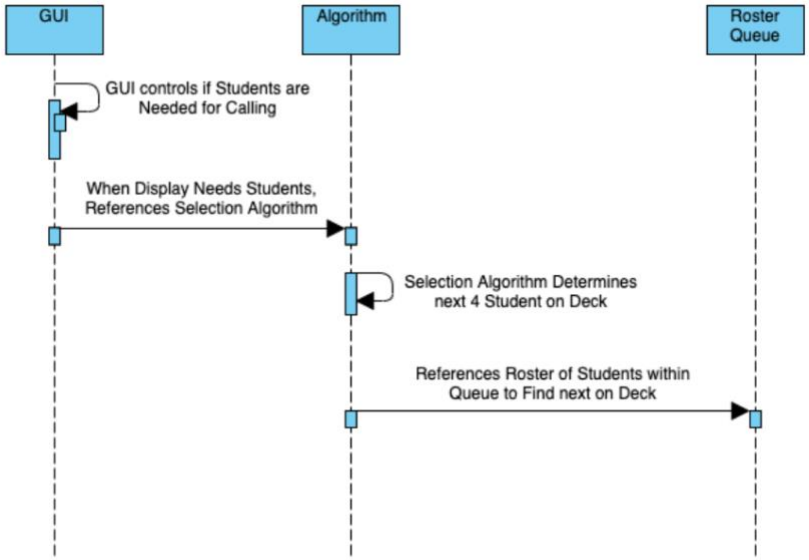
   Having direct communication with the Teacher Interface rather than passing through the Action module.

5. Dynamic Models of Operational Scenarios (Use Cases)

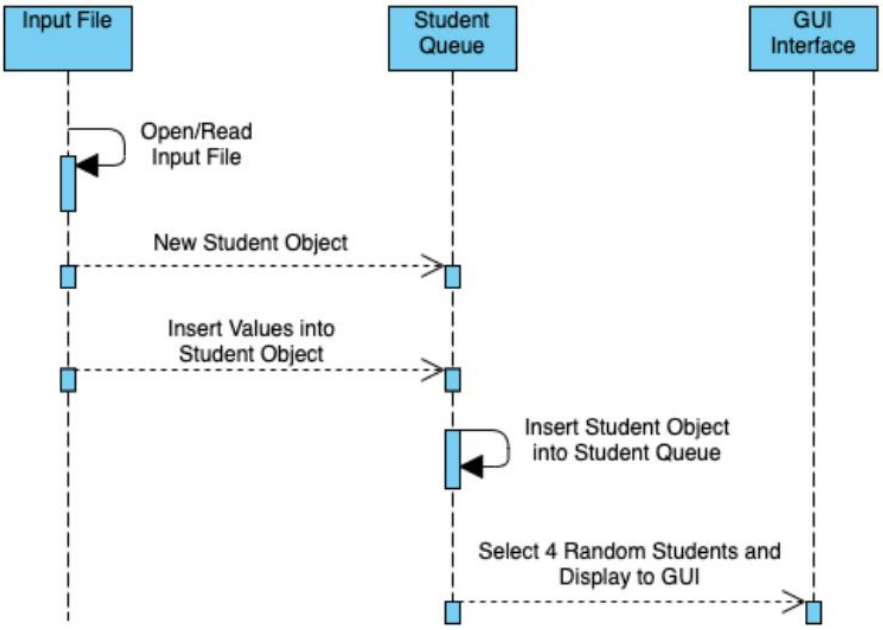
5.1 Summary Performance Model



5.2 Day-to-Day



5.3 Input File Model



6. References

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

7. Acknowledgements

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.

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.