Software Requirement Specification (SRS)
for a Classroom Cold-Call Assist Software Program
A. Hornof - January 10, 2020 - v2

1. System Summary

The system is intended to assist an instructor with "cold calling" on students in a classroom. "Cold calling" means requesting input from a specific student who did not raise his or her hand. At the very least, the system will provide the instructor with the names of randomly-chosen students in the class so that the instructor can direct the next question to those students. The system is designed to equally distribute the cold calls across all students, across multiple days. The system provides a compact (in terms of screen real estate) list of the students of who are next in the queue, which can be displayed to the class, ideally across the top of the slides being displayed, so that students can "warm up" to being called on. The system will keep track of all the students who were permit the instructor to discretely "flag" students that the instructor might want to follow up with after class. The instructor can use a handheld physical push-button device, such as a clicker or a Wii, to remove

2. The Motivation

The system is partly motivated by the Bowles (2019) "Human Contact Is Now a Luxury Good" article that appeared in the *New York Times*. (https://www.nytimes.com/2019/03/23/sunday-review/human-contact-luxury-screens.html) The article points to the logical conclusion that human-to-human classroom instruction may largely disappear in the next few decades. The cold-call-assist program attempts to maximize the human contact in classroom instruction while we still have it.

The system is partly motivated by Professor Kingsfield of the *Paper Chase* television show from the 1970s. Note the photo roster that the professor references 8 seconds into this video: https://www.youtube.com/watch?v=WCkMDvikGNM

The system design, especially the ideas for how to "warm up" the cold calls, is influenced by Dallimore et al. (2006). Nonvoluntary Class Participation in Graduate Discussion Courses: Effects of Grading and Cold Calling. Journal of Management Education, 30(2), 354-377, https://journals.sagepub.com/doi/abs/10.1177/1052562905277031.

---

3. Use Cases

A. Day-to-day in-class usage
Preconditions: An instructor wants to encourage students to participate in class discussions, is comfortable "cold calling" on students, wants to be egalitarian (treat students equally) with her cold-calling, and uses a computer as part of her in-class presentations.

1. The instructor arrives in the classroom for a lecture, connects her computer to the projector, opens Powerpoint, greets the class, announces the lecture topics, asks if anyone has any questions, and starts lecturing from the slides.
2. After lecturing for five minutes, the instructor wants to check in and see if students are understanding the material, and so she poses a question to the class. She waits ten seconds but nobody answers.
3. The instructor recalls that she has the cold-call-assist software installed on her laptop, and so she searches her hard drive for the program (such as, on a Mac, by typing command-space and the name of the program). This entire step takes less than 5 seconds.

4. A list of four student names appears. They are "on deck" (as in baseball). Ideally, it is automatically positioned so that it minimally blocks the PowerPoint slide content, such as a horizontal list of names at the top of the display. The instructor switches the focus of the operating system back to the Powerpoint presentation, but the name window stays in front.
5. The instructor notes that none of these students were cold-called at the beginning or the end of the previous lecture, reassuring her that the system re-jumbles the names between runs.
6. The instructor asks if any of the students in the list would like to answer the question. None volunteer, and so she cold calls on the third student in the list.
7. After the student responds, the instructor thanks the student and removes the student from the list in one of the following two manners:
   A. Arrow Key Input (preferred).
   (a) Press the left and right arrow keys to highlight the student's name in the list.
   (b) Press a down arrow key to "drop" the name from the visible list without a flag, or
   (c) an up arrow key to remove the name but also "raise" a flag for that student.
   B. Numeric Key Input:
   (a) Press the number key *1* through *4*, whichever corresponds to the position of the student on the list.
8. The fourth student on the list moves into the third position. A new student's name appears in the fourth position.
9. After class, the instructor reviews a list of students who responded to cold-calls that day, and notes a couple students who were flagged for one reason or another. She uses her mouse to copy from a list of preformatted strings of "Firstname Lastname <flast@uoregon.edu>" that she can copy into the "To:" header of an email, and email encouragement to that student.

Postconditions: The system, though it is not running, is completely ready for its next usage.

B. After class, the instructor reviews if any students might benefit from encouragement.
Preconditions: After class, in her office, the instructor knows that she did some cold-calling in class that day, and that a student demonstrated special interest in a topic. She wants to email a conference paper on the topic to that student.

1. After class, in her office, the instructor opens the daily log file and reviews its contents.
2. She sees the heading at the top of the file indicating that this is the daily log file for the cold-call-assist program.
3. She sees today's date immediately under that heading, and then one line for each cold-call that she made that day. Each line is formatted as follows:
   <response_code> <tab> <first name> <last name> "<" <email address> ">"
   The <response_code> is blank if there was no flag, and "X" if there was. Such as:
   X Fatima Patel <fpatel@uoregon.edu>
4. The instructor recalls that Fatima was breakdancing in class and did not hear when the instructor called on her, and wants to email a polite request to the student to practice her headspins outside of class. Or the instructor recalls that Fatima showed special interest in a topic, and wants to email a conference paper.
5. The instructor drags her mouse cursor across the portion of the line that includes the student's name and email address in brackets, issues a "copy" command (such as using the right mouse-button), and then pastes the string into the "To:" header of her email client.
6. She composes and sends the email.

Postconditions: The instructor is done with her after-class review. Students have been emailed.

C. Adjust the contents of the system a week into the term.
Preconditions: The instructor needs to update the contents of the roster that is stored in cold-call-assist but does not want to loose any of the data she already entered into the system.

1. The instructor exports the cold-call-assist roster into a tab-delimited file that is correctly formatted to be reloaded into the system later. She is confident that the exported file will not overwrite an existing file.
2. The instructor loads the tab-delimited file into Excel, adds some phonetic spellings based on what she learned from students in class, and exports from Excel into a tab-delimited file.
3. She issues a command to import the file into cold-call-assist. The system warns her that there is already data in the system and, before overwriting the data in the system, lists the students whose data will be changed in any manner (such as dropped from the roster, added to the roster, or any other change at all).

Postconditions: The instructor is ready for the next day of class, with current data.

D. At the end of the term, the instructor reviews a summary of class participation

Preconditions: The instructor wants to review a summary of each student's performance across all of the times that each student was cold-called during the term.

1. The instructor opens the summary performance file and reviews its contents.
2. She sees the heading at the top of the file indicating that this is the summary performance file for the cold-call-assist program, and headings for the "columns" in the tab-delimited file.
3. She sees a list of students with performance data after each student, each line formatted as: <total times called> <number of flags> <first name> <last name> <UO ID> <email address> <phonetic spelling> <reveal code><list of dates> with a tab between each field, and a Unix linefeed at the end of the line. The list of dates includes all the days the student was cold-called, are formatted as YY/MM/DD, and are in chronological order.
4. The instructor imports the data into a spreadsheet and figures out an Excel formula that uses the two numbers at the start of each line to compute a cold-call participation score for each student. Now there are three numbers related to cold calls for each student.
5. The instructor imports all three numbers into Canvas so the students can see their cold-call performance for the term.

Postconditions: The instructor is done using the system for the term. Students have received feedback, the rewards of their preparation for being called-on in class.

E. Preparation at the start of the term.

Preconditions: The instructor, working in her office, prepares for the first day of class.

1. The instructor logs on to Duckweb.
2. The instructor downloads the student roster and reformats it into a tab-delimited file with a separate field for first name, last name, UO ID, and email address.
3. The instructor opens up the student photo book for the class, and drags each photo to a "Photos" folder on her Macintosh. As she drags each photo, she replaces each filename with the student's UO ID number, such as "950123456.jpg".
4. The instructor starts up the cold-call-assist program and does an initial import of the data into the system.
5. The instructor reviews the data in the system for accuracy, reviewing students in alphabetical order by last name, either by flipping through them one at a time or by reviewing a photo roster that lists many students per page.

Postconditions: The instructor is ready for the first day of class.

---

4. Instructor-System Interactions

4.1. Call on students and remove them from the "on deck" list.

There are two possibilities here. One or both could be implemented. Option *A* is preferred because it is expected to require less time looking at the computer to find the keys *1* through *4*.

Option A. Arrow Key Input (preferred).

The instructor can call on any student in the visible queue, and remove that student from the queue by pressing the left and right arrow keys. As soon as the first arrow key is pressed, some sort of simple highlighting will appear such as by simply inverting the text (to white on black).

If the left arrow key is pressed, the highlighting starts at position 1.
If the right arrow key is pressed, the highlighting starts at position 4.

For example, if the following students are currently visible in the queue, and the left arrow key is pressed, the highlighting starts like this:

Maria Diego Anna Cavendish Yunfeng Zhang Aniyah Jackson

Pressing the right-arrow key moves the highlighting to the right, and it does not cycle back around to position 1. Pressing the down arrow key removes highlighted name from the list, and pushing the up arrow key removes it and sets a flag for that removal instance.

All of the names on the list shift to the left, and a new name appears at position 4.

Option B. Numeric Key Input

The instructor can call on any student in the visible queue, and remove that student from the queue by pressing a key that corresponds to that position on the list. For example, if the following students are currently visible in the queue:

Maria Diego Anna Cavendish Yunfeng Zhang Aniyah Jackson

and Anna raises her hand and answers a question, the instructor can press the *2* key to removed from the list. Yunfeng will move to the second position, and a new name will appear in the fourth position.

If a <flag key> is pressed within one second of pressing a button to remove a student from the front of the queue, the entry in the daily log file for this flag is annotated in the daily log file as described in Use Case B, above. This will facilitate, for example, the instructor sending email a student after class.

<flag key> includes *all* of the following keys: <Q> <W> <E> <R>

Any time a button is pressed to remove a student from the queue, that removal triggers the addition of an entry into the daily log file as described in Use Case B, above, and the term log file as described in Use Case D, above.

4.2. PowerPoint is the active application but the Cold Call window is the foreground window.

In order for the instructor to have full control over course presentations (PowerPoint, Firefox, etc.), the Cold Call system must sit in the background. However, for its window to be seen, it must sit in the foreground of these other applications. Through all of this, Cold call also needs to be listening for the keystrokes specified above, even though it is the background application.

4.3. Display

A compact horizontal list of names that appears on the screen above lecture materials.
The very top of the window showing the student names should be configured as follows:

Next students: <first> <last> <first> <last> <first> <last> <first> <last>

This way, provided that the very top of the display is visible to students, everyone can see who is up next and who is “on deck”.

4.4. Roster Input and Output

The system will have a command to import a file, and a command to output a file. The system will confirm that the file is in the format specified below and, if it is not, the system will (a) not read in the file and (b) output a useful error message explaining exactly what the problem is, and what the user needs to do to fix the format.

4.5. Change-related requirements

It should be possible to change the system's use of tab-delimited files to a use of comma-delimited files by modifying only one line of source code, and by converting each tab to a comma in each data file.

All numerical parameters, and all keystroke assignments, discussed in the specification must be set at the top of a source code file, and easily changeable by a programmer during development and maintenance of the code.

There should an easy way to change all key mappings. At the very least, they should be clearly specified in one location near the top of a source code file. Perhaps an even more flexible solution would be to have the key mappings read-in from a text file that includes a comment at the top of the file explaining the purpose of the file. (The system should still run if this file is missing.)

---

5. Data Constraints

5.1. The Ordering of Students in the Queue

The ordering of the students will be randomized.

After a student is called on, that student will be removed from the front of the queue, and re-inserted into the queue, but not in the *n* % of the list that at the front of the queue, to delay the next on-deck appearance. The *n* parameter will be set in a single location at the top of a source code file, and will be set such that many runs of the program will result in an equal distribution of cold-calls across all students.

The queue may be maintained across different runnings of the program (that is, after the program is quit and restarted). The queue should therefore be saved and reloaded. Optionally, at startup, the front *n*% of the queue could be re-randomized, separately from the back 1-*n*% of the queue.

Every student in the roster should always be somewhere in the queue, though usually only the front of the queue will be visible on the display.

The system will run without crashing with any number of students in the roster.

Student ID numbers should never be displayed on the screen.

5.3. Realistic Sample Data

The developers must provide sample data (such as initial roster files to read into the system, and properly-sized photos) to simulate actual usage of the system. The data should be realistic based on characteristics of classes at the University of Oregon. The system must ship with realistic sample data to test the system.

5.4. The user names will reside in a roster file with the following format:

<first_name> <tab> <last_name> <tab> <UO ID> <tab> <email_address> <tab>
<phonetic_spelling> <tab> <reveal_code> <LF>

The UO ID will be nine digits.
The <reveal_code> will be used to indicate details about this entry, such as if the photo will be displayed to other students in the classroom.
<LF> is a Unix line feed character.
The spaces around the <tab> and <LF> characters should not be added to the file.

The first line of the file, up until the first <LF>, will be a comment that is not modified by the system, both when reading from and writing to the file.

5.5. Random Distribution Verification Mode
Restated: There should be a straightforward but perhaps slightly hidden way to test the randomness of the system, such as a keystroke combination that is specified in the system documentation, that automatically restarts the program 100 times, and issues 100 random cold calls each time the program is running. The data should be pooled in the output data file, to be analyzed to evaluate system performance. The user should be given the option of quitting out of the test before starting it (with a warning that the output data file would be overwritten, if that would be the case).

---

6. System Startup and Shutdown

6.1. System Startup
It must be possible to start the system by double-clicking on an application that has a unique application name. That is, it cannot require the opening of a Terminal window and the typing of a command at the command line.

The system must start up within one second of double-clicking on the icon.

At startup, and while using the system, the system should not generate any error messages that interfere with the running of the program, such as by putting a "modal" dialog box that requires user input to proceed. The system will always forge ahead as best as it can.
For example:
(a) If, during execution, the program can suddenly no longer read from disk, the program will work with a copy of the roster that was already loaded into memory.
(b) If, during execution, the program can suddenly no longer write to disk, such as if the disk is full, the program will still provide a continuous random list of names for cold-calling but not write any logging information to disk.
(c) If the program cannot find an file that is expected to exist, such as to append data to that file, the program will create a new file with an appropriate name and use that file, but will never overwrite an existing disk file.

At system startup, the random number generator will be re-seeded with a different seed.

6.2. Exiting the System
Though no data should ever be lost, or overwritten on disk, the system should never interrupt the closing of the program after the user issues a command to quit the program. This could be accomplished in part by always saving to file after every keystroke.

7. Optional Secondary System: A Photo-Based Name-Learning System

This is a separate, minor add-on project that could also be completed.

The basic idea is to provide a system that assists in using student photos to learn students' names.

The display would show many (or all) student photos, and provide an easy way to look at a photo, not see the name, say the name, and then see the name with as little hand movement as possible; this could even be accomplished by having the name in a small (user settable) typeface that would require an eye movement in order to read the name. It is okay to rely on the user's self-discipline when the user is using the program.

There should also be a means of separating photos of names that have been learned, from photos of names that have not been learned, so the user can focus his or her attention on the names still to be learned. Such as how you remove cards from a stack of study cards as you learn the content of each card.

Associating names with photos could be done simply by using file names that are <FirstName> <Lastname>.jpg.

7.1. Name and Photo Learning Mode

The system could also have another means of assisting the instructor in learning the names of students based on their photos. This could be in the form a "flash card" game that proceeds as follows:

1. A randomly-chosen student's photo is displayed.
2. The instructor says aloud the name of the student.
3. She presses a button to display the name and phonetic spelling.
4. She decides if she was right, and presses one buttons if yes, and another if no.
5. If she was right, that name and photo are removed for the remainder of the game.
6. The game ends when the instructor has pressed the "correct" button for every student.

The ability to recall a name from a photo could be enhanced by showing small photos next to the names of students who are in the queue.

7.2. Photo Files

The system should accommodate a photo size that is easy to modify in the software settings, and is initially set to a fixed height, set in pixels, and accommodates a photo with a width that is between 50% and 100% that of the height

The default size of the photo that is input will initially be set to 200 pixels.

If the system includes photos, these will be read from a single directory, which could be anywhere on the disk.

There will be a means to view the photos in alphabetical order, or re-arranged randomly.

8. Build-Related Constraints

8.1. Target Platform
The system must run on Macintosh OSX 10.13 (High Sierra) or 10.14 (Mojave).

8.2. System Document File Formats
All system-related and system-development-related documents that are intended for human reading must be in either plain text or PDF. For example, Microsoft Word, Microsoft Excel, or markdown language documents must be converted into plain text or PDF.

8.3. Programming Constraints
- The system may be built in C/C++, the C++ standard library, Cocoa, and no other components. (Note that an XCode command line tool could fulfill many of the requirements.)
- The system may be built using Python 3 along with The Python Standard Library https://docs.python.org/3/library/index.html, but no other imports except for pyHook.
- The system may be built using Java along with Java Standard Edition modules https://docs.oracle.com/en/java/javase/12/docs/api/index.html , but no other imports.
- C++ code must comply with C++11.
- Python code must run in Python 3.4, 3.5, 3.6, and 3.7.
- Java code must run in Java 7 or 8.
- Instructions must be provided for how to compile the code.
- No server connections may be required for either installing or running the software.
- No virtual environments may be used.
- No gaming engines such as Unity may be used.

8.4. Installation
- There can be at most 10 steps to compiling the code and running the program.
- An experienced computer programmer should not require more than 30 minutes working alone with the submitted materials to compile and run the code.