

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

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

January 14, 2020 – v0.1

## **Table of Contents**

- 1. SRS Revision History 2 check page numbers - FIXME**
- 2. The Concept of Operations (ConOps) 2**
  - 2.1. Current System or Situation 2**
  - 2.2. Justification for a New System 2**
  - 2.3. Operational Features of the Proposed System 3**
  - 2.4. User Classes 3**
  - 2.5. Modes of Operation 4**
  - 2.6. Operational Scenarios (Also Known as “Use Cases”) 4**
- 3. Specific Requirements 8**
  - 3.1. External Interfaces (Inputs and Outputs) 7**
  - 3.2. Functions 7**
  - 3.3. Usability Requirements 7**
  - 3.4. Performance Requirements 7**
  - 3.5. Software System Attributes 8**
- 4. References 8**
- 5. Acknowledgements 9**

# 1. SRS Revision History

Date	Author	Description
4-8-2019	ajh	Created the initial document.
1-14-2020	bvm	Wrote rough drafts to all sections. Added FIXMEs to revisit sections.

Add here

## 2. The Concept of Operations (ConOps)

### 2.1. Current System or Situation

“Cold calling” is a system that requests input from specific students who did not raise their hand. This system provides a way to randomly-choose students to participate in class by asking questions or making comments. The instructor may use “cold calling” to increase participation in a class and hear from students that may otherwise not participate often or at all. Throughout multiple days, this system equally distributes cold calls to all students in the class. As well as having the base system of selecting students randomly to participate, there is always a list of students who are next in line to be cold called. This list allows students to “warm up” to being called on by knowing they must participate soon.

### 2.2. Justification for a New System

Currently, there is no program that allows cold calling to be done without the professor developing a way to randomly select students as well as announce or write who is next in the queue. This can interrupt valuable class time and distract from the topic at hand. This can make it difficult or less appealing for the instructor to want to implement cold calling in their class. But, in an article titled “Impact of Cold-Calling on Student Voluntary Participation,” written by Dallimore, Hertenstein, and Platt in 2012, cold calling is researched and found to be highly effective to increase voluntary class participation

([http://ecommons.med.harvard.edu/ec\\_res/nt/B5F7F484-6BCB-4DC3-8B5A-76AB2D53FB4B/Cold-calling.pdf](http://ecommons.med.harvard.edu/ec_res/nt/B5F7F484-6BCB-4DC3-8B5A-76AB2D53FB4B/Cold-calling.pdf)). Increasing class participation is very beneficial to students and instructors by both parties getting more out of each lecture. Active lectures are

becoming more important as many students do not get as much out of non-participating lectures as discussed in “Should Colleges Really Eliminate the College Lecture?” written by Gross-Loh in 2016 (<https://www.theatlantic.com/education/archive/2016/07/eliminating-the-lecture/491135/>).

So, class participation is very important and cold calling increases voluntary, positive participation. There is no cold calling system that is easily available to instructors. Instructors may be discouraged from implementing a cold calling system because of the work that goes into it manually.

### **2.3. Operational Features of the Proposed System**

The proposed system will provide instructors with a simple way to integrate cold calling into their classroom. This system will provide a list of randomly generated names from the class roster so the instructor can easily see and display who is in the queue for cold calling. Whichever student from the queue that participates may be selected by input from the instructor, such as the arrow keys, and either flagged or simply “dropped” from the queue. After the student’s name is gone from the queue, another one fills in the end. This cycles through the whole class roster until all students have been through the system, and continues through the roster again in a random order. Along with the direct use of cold calling, the instructor may also open a daily log file from the lecture of the class cold calling was used in. This log would have the date of when the system was used as well as one line for each cold call that they made that day. Each line has the response\_code, first name, last name, and email of the student. The response\_code is simply marked with an “X” if the instructor flagged that student during the time the student was cold called. In this daily log, the student may copy the email or take notes from flagged students.

Lastly, the proposed system includes a summary performance file of the term of cold calling. This can be viewed to see each student’s participation. Each line in the file contains the total times the student was called, number of flags the instructor gave in their cold calls, the student’s first and last name, UO ID, and email address, the phonetic spelling of their name, reveal code, and a chronological list of dates that they were cold called. This summary may be exported into a spreadsheet.

### **2.4. User Classes**

The user classes for this system include instructors and students. The instructor uses the cold calling system in lecture to promote participation in class while not having to manually set up a cold calling system. They also interact with it for reviewing students' participation. Students may use it by being able to view who is next in the queue if the instructor is presenting their screen to the class. Programmers are also interacting with this system after it is built for maintenance and support.

## **2.5. Modes of Operation**

Different modes of operation for this system include user, instructor, administrator, training, and maintenance. User would include admin and instructor, where each of those are just different uses of the system. Administrator mode could view the system in the regular use mode (instructor mode) or a training mode. Instructor mode could use the system as it's intended for day to day use. Maintenance mode is different mode for the programmers to edit and view the system.

## **2.6. Operational Scenarios (Also Known as "Use Cases")**

**Use Case: An instructor wants to cold call students in class.**

### ***Brief description:***

This use case describes an instructor cold calling a student by using the cold calling system as intended for its day-to-day use.

**Actors:** An instructor.

### ***Preconditions:***

1. The instructor has the roster of the students in the class in the cold calling system.
2. The instructor uses a computer during lecture that they present or show resources on.
3. The instructor flags some students during the cold calling process.

4. - FIXME what else here

### ***Steps to Complete the Task:***

1. The instructor arrives to class, connects their computer to the projector, starts up the cold calling software, and starts class as usual. The powerpoint presentation and other resources the instructor is

using do not overlay the cold calling software. The software remains in the foreground as lecture continues so students can see who is in the queue to be cold called.

2. After reaching a point in the lecture that the instructor wants the class to participate, they look at the cold calling software to see who is in the queue to be cold called.

3. A list of four students is available to the instructor on the software. These names are horizontally listed in a small window at the top of the instructor's screen. The instructor knows that none of the students in the list were cold called in the previous lecture, which shows that the software is randomizing names as well as not reusing previously used names before all names are cold called.

4. The instructor asks if any students in the queue would like to answer, and the second student in the queue participates.

5. After the student contributes to lecture, the instructor removes the student from the queue by:

- a. pressing a left or right arrow key to highlight the student's name in the list.
- b. Press the down arrow key to "drop" the student's name without a flag, or
- c. Press the up arrow key to remove the student's name while also adding a flag.

6. The student's name is removed, and the third and fourth students' names move into the second and third positions respectively. A new student's name appears in the fourth position.

7. This cycle continues until the class is complete. After class, the instructor wants to send an email to the students who were cold called that day and were flagged. The instructor copies a preformatted string ("firstName lastName <flast@uoregon.edu>") that they then can paste into the "To:" of an email client.

***Postconditions:***

The instructor is ready for the next class because the system is completely ready for its next usage.

**Use Case: After a class, an instructor wants to email the students they flagged with an article about how to be respectful in class.**

***Brief description:***

This use case describes an instructor viewing the daily log file to find a set of students they want to send an email to.

**Actors:** An instructor.

***Preconditions:***

1. The instructor has had class and flagged students in that class.
2. - **FIXME** what else here

***Steps to Complete the Task:***

1. The instructor opens the cold calling software and opens the daily log file to review who was flagged. At the top of the daily log file, there is a label indicating that it is the daily log file.

2. Under the heading “Daily Log,” there is the date. After the date, there is one line for each cold call that happened that day. Each line is formatted as:

`<response_code> <tab> <first name> <last name> “<”<email address>”>`

The response\_code is either empty if the cold call wasn’t flagged, or an “X” if it was flagged.

3. The instructor looks through the lines of the file to find the students they flagged for being disrespectful during the cold calling process.

4. The instructor then positions their mouse over the particular cold call line that they want, and copies (such as right-clicking and selecting “Copy”). The instructor then goes to their email client and pastes into the “To:” section.

5. The instructor copies whoever else they want to email and pastes their emails as well, writes the email, and sends it.

***Postconditions:***

The instructor has emailed the problem students and is complete with their tasks.

**Use Case: The instructor wants to reward who participated most and punish who participated least in their class at the end of the term.**

***Brief description:***

This use case describes an instructor viewing a summary of all class participation.

***Actors:*** An instructor.

***Preconditions:***

1. The instructor has used the cold calling software throughout the term or over an extended time.
2. The instructor flagged people who did very well when cold called and who were problematic.

***Steps to Complete the Task:***

1. The instructor opens the summary performance file from the cold calling software.
2. Under the heading "Summary Performance," there are headings for columns related to the data within each row beneath it. Each line is 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 delimiting each field and a Unix line feed at the end of the line. The dates are the dates the student was cold called in chronological order and formatted as YY/MM/DD.
3. The instructor imports the data into a spreadsheet and uses the spreadsheet software to compute who raised the most flags per cold call. Since the instructor marked good and bad instances of the cold call, they look through and see who sticks out most in their mind to be the most beneficial in participation and who was the most problematic.
4. The instructor then gives extra credit to the best student and emails the bad student about a meeting to discuss their overall term behavior.

***Postconditions:***

The instructor has emailed the problem student and gave extra credit to

the good student. The instructor is complete with their tasks.

### 3. Specific Requirements

Requirements should be prioritized, with each classified as (a) must have, (b) should have, (c) could have, and (d) won't have. These can be recalled with the memory aid of MoSCoW (vanVliet 2008, p.237). When reading requirements, it should be very easy to see how each requirement is classified, such as by having them grouped by priority. - help on interpreting this? FIXME

#### 3.1. External Interfaces (Inputs and Outputs)

##### Input: Arrow Keys

**Description:**

To collect input for highlighting names and removing them.

**Source of input:**

The user pushing the keyboard's arrow keys.

**Valid ranges of input:**

The up, down, left, and right arrow keys.

**Units of measure:**

FIXME

**Data formats:**

FIXME

**FIXME: Add this if it's an input:: " Input: Double-clicking application icon"**

##### Input: Class roster file

**Description:**

A tab-delimited file with the first name, last name, UO ID, email address, phonetic spelling, and notes of all the students in the class to upload data into the system.

**Source of input:**

The user.



***Valid ranges of input:***

.txt files.

***Units of measure:***

MB. **FIXME**

***Data formats:***

The file must be tab-delimited and formatted as:

<firstName> <tab> <lastName> <tab> <UO ID> <tab> <email address>  
<tab> <phoneticSpelling> <tab> <reveal\_code> <LF>

The file must include a header line that will be treated as a comment until a <LF> is hit.

**Output: The display of the cold calling system**

***Description:***

A compact horizontal list of names in a window that stays in the foreground of other applications. It will still listen to the keystrokes needed in order to use the cold calling system.

***Source of output:***

The user's computer screen.

***Valid ranges of output:***

A window displaying names and a small menu with options such as "Performance Summary."

***Units of measure:***

**FIXME**

***Data formats:***

**FIXME**

**Output: Class roster file with cold calling information**

***Description:***

A tab-delimited file with the first name, last name, UO ID, email address, and cold call system information of all the students in the class.

***Source of output:***

a txt file in a location chosen by the user.

***Valid ranges of output:***

.txt files. **FIXME**

***Units of measure:***

The file must also not exceed 1000 lines. (does this need to be MB or something???)

**Data formats:**

The file must be tab-delimited and formatted as:

<firstName> <tab> <lastName> <tab> <UO ID> <tab> <email address>  
<tab> <phoneticSpelling> <tab> <reveal\_code> <LF>

### 3.2. Functions

#### Input: Arrow Keys

**Validity checks on inputs:**

Ensure the input is mapped to the keycode of the arrow keys.

**Sequence of operations in processing inputs:**

Take the input, determine which arrow key it was:

- If the left arrow key is called, the highlight starts at position 1. After a name has been highlighted, pressing the left arrow key will move the highlighted selection to the left until it hits the first position.
- If the right arrow key is pressed, the highlighting starts at position 4. After a name has been highlighted, pressing the right arrow key will move the highlighted selection until it hits the fourth position.
- If it is a down stroke and a name is highlighted, remove the name from the current queue, add it to the “not yet seen” queue behind the current list of students who haven’t been called this round yet. Add a new name from the current queue into the “on deck” list on screen.
- If it is an up stroke, remove the name from the queue and add it to the “not yet seen” queue. Along with removing it, add the “X” to the student’s information in the <response\_code> area. Add a new name to the on screen queue.

**Responses to abnormal situations:**

- If a down or upstroke is inputted, but no

name is highlighted, nothing should occur.

- If a name is highlighted in either position 4 or position 1 and the user tries to go past this point in the respective directions (e.g. keep going right when the highlighted name is already in position 4), nothing should occur.

- FIXME - any other ideas?

### ***Relationship of outputs to inputs:***

#### ***(a) input/output sequences:***

- The input of pressing an arrow key to highlight a name should produce a highlighted name in the display output.
- If the user intends to remove a name by pressing up or down, the name should be removed, all the names should shift to the left, and the fourth position should fill with a new name.

#### ***(b) formulas for input to output conversion:***

- The keystroke should be taken into the software and raise an event that calls the function to handle the display. FIXME

**Input: Double-clicking application icon (FIXME: is this an input?)**

### ***Validity checks on inputs:***

Ensure the system is on disk and information loads up fine.

### ***Sequence of operations in processing inputs:***

The user double clicks the icon. The display should pop up within one second and be running and ready to use.

### ***Responses to abnormal situations:***

- If, during execution, the program cannot read from disk anymore, the

program will work with a copy of the roster that was loaded into memory.

- If, during execution, the program can suddenly no longer write to disk, the program will still provide a continuous random list of names for cold calling but not log anything onto the disk.

### ***Relationship of outputs to inputs:***

#### ***(a) input/output sequences:***

The user double clicks the icon and the system starts up.

#### ***(b) formulas for input to output conversion:***

- Once the system know to start up, it should load all the saved data from the last session.

### **Input: Class roster file**

#### ***Validity checks on inputs:***

Ensure the file type is the type intended to be inputted. Check to ensure file isn't too large. Also, once the base file is checked, the size of each field must be checked. For example, the UO ID number must be nine digits. Along with the size of the field, the correct formatting of the fields should exist (as discussed above).

#### ***Sequence of operations in processing inputs:***

Have the user click a button to insert a roster. This should prompt the user to browse for a file to upload. Once a file is selected, the user should click submit. The system will review if the file is the right type, and under a certain size. If it is, allow input and start parsing the data into our system. The data of the students should then be available to view and interact with.

#### ***Responses to abnormal situations:***

- If the user tries to upload too large of a file or the incorrect file type, the request will be denied and the user will be given a reason as to why they cannot use that file.
- If the system is importing what seems to be a correct file that ends up not containing the correct information, the system should cancel the import and give an error message on the fields within the file.
- If the system detects the file isn't in the right format, an error message should be given and the import process should stop.

***Relationship of outputs to inputs:***

***(a) input/output sequences:***

If the user uploads a file, the data should be represented and stored in the cold calling system. The data should be outputted on the display.

***(b) formulas for input to output conversion:***

- Once the file is inputted, our system will parse the data and store it in a data structure that best fits our needs. The system then runs the display with the new data and contains the queue with the new list of students.
- If the user wants to export the data, the user clicks on some sort of export button and decides where to save it. The data is then compiled from our data structure into a tab-delimited file and saved.

**Output: The display of the cold calling system**

***Validity checks on output:***

Make sure the screen size isn't too small for the window the system is

trying to display. Check to see if the program is running already.

***Sequence of operations in processing outputs:***

The display will listen for events to change itself. For example, if an event is fired, such as an arrow key pressed, the correct function will be called to handle that event and to update the display.

***Responses to abnormal situations:***

**FIXME**

***Relationship of outputs to inputs:***

***(a) input/output sequences:***

- The display contains buttons that may be pressed to export or import data. If this is the case, the correct input and output actions must be called.
- If the startup icon is clicked, the display should pop up within one second.

***(b) formulas for input to output conversion:***

- If a button is pressed, raise an event that then gets handled by the respective input/output.
- Always listen for events for input to update the display.
- If the icon to start the cold call system is double clicked, start the display and system up.

**Output: Class roster file with cold calling information**

***Validity checks on outputs:***

Ensure that there is space on the user's computer to output the file.

### ***Sequence of operations in processing outputs:***

If the user clicks export the class roster with cold calling information, they should then select where they want to save the file. The system then takes the data and formats it into a .txt file that is tab-delimited.

### ***Responses to abnormal situations:***

FIXME

### ***Relationship of outputs to inputs:***

#### ***(a) input/output sequences:***

The display export button must be clicked on in order to export the file. Once the export is clicked, the user determines where they want to save the file. Once that is determined, the user can see on the display if their file was exported.

#### ***(b) formulas for input to output conversion:***

- If the button is clicked on the display to export, an event should be raised to handle that the user wants to save the file. This should then call the function that saves the file as an output.

## **3.3. Usability Requirements**

FIXME - idk lol I just wrote stuff

The cold calling system should be random each time it's loaded up and retain the data from the previous session. The cold calling system at its base should have a queue of four visible, randomly chosen students that are then highlight-able and removable with the arrow keys. The user should be able to import and export tab-delimited files to and from the system. A Daily Log should be available to the user and contain students who were cold called that day and list their information

such as their names, emails, and flag status based on that call. A summary of an extended period of time should also be available to the user. This summary will include more data on how many cold calls students were involved in and how many of those were flagged, along with their names, emails, notes on the student, phonetic spelling, and UO ID. The display of the system should not inhibit much of the screen and it should be in the foreground view of the screen. Lastly, it also should be straightforward in its use.

### **3.4. Performance Requirements**

FIXME- need help here idk either

The system should start up when the icon is double-clicked within one second.

Specify the static and the dynamic numerical requirements placed on the software or on human interaction with the software. For example: (a) Static numerical requirements may include the amount and type of information to be handled. (b) Dynamic numerical requirements may include the amount of data to be processed within certain time periods.

Performance requirements should be stated in measurable terms. For example,

“95% of the transactions shall be processed in less than 1 second” rather than “An operator shall not have to wait for the transaction to complete.” (ISO/IEC/IEEE 29148:2011)

### **3.5. Software System Attributes**

FIXME

Specify the required attributes of the software product, such as reliability, security, privacy, maintainability, or portability. (ISO/IEC/IEEE 29148:2011) Review a comprehensive list of software attributes, or software qualities, such as are provided in van Vliet (2008) Chapter 6. Decide on a relatively small number of attributes that are most important for this system. Explain why each attribute is important, and what steps



or plan will be taken to achieve those attributes. This could include constraints on attributes of the system's static construction, such as testability, changeability, maintainability, and reusability. (Faulk, 2013)

## 4. References - **FIXME do I keep his references in here?**

This section lists the sources cited in the creation of this template document. An SRS should reference all of the sources that it draws from. If sufficient citations are provided "in line" (at the point of reference) in the document, this section may not be necessary.

IEEE Std 1362-1998 (R2007). (2007). IEEE Guide for Information Technology—System Definition—Concept of Operations (ConOps) Document. <https://ieeexplore.ieee.org/document/761853>

IEEE Std 830-1998. (2007). IEEE Recommended Practice for Software Requirements Specifications. <https://ieeexplore.ieee.org/document/720574>

ISO/IEC/IEEE Intl Std 29148:2011. (2011). Systems and software engineering — Life cycle processes — Requirements engineering. <https://ieeexplore.ieee.org/document/6146379>

ISO/IEC/IEEE Intl Std 29148:2018. (2018). Systems and software engineering — Life cycle processes — Requirements engineering. <https://ieeexplore.ieee.org/document/8559686>

Faulk, Stuart. (2013). *Understanding Software Requirements*. [https://projects.cecs.pdx.edu/attachments/download/904/Faulk\\_SoftwareRequirements\\_v4.pdf](https://projects.cecs.pdx.edu/attachments/download/904/Faulk_SoftwareRequirements_v4.pdf)

Oracle. (2007). White Paper on "Getting Started With Use Case Modeling". Available at: <https://www.oracle.com/technetwork/testcontent/gettingstartedwithusecasemodeling-133857.pdf>

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

## **5. Acknowledgements**

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 1362-1998 and ISO/IEC/IEEE Intl Std 29148:2018.