CIS 422 Project 1: Ducks on Deck Project Plan

Luying Cai (lc), Ellie Kobak (erk), Kalyn Koyanagi (kek), Liza Richards (ljr), and Kelly Schombert (ks)

January 8, 2022 – v1.0

Table of Contents

1. Project Plan Revision History

Date	**Author**	**Description**
1-8-2022	ks	Created the initial document and wrote first draft to be proposed at second group meeting.
1-10-2022	ks	Revised Organization and Roles section and Build plan section

2. Management Plan

2.1. Organization and Roles

Each group member is assigned the following role which includes the outlined responsibilities:

- Writing lead: Liza Richards

    - The writing lead will verify the quality and completion of the writing in project documents such as the SRS and SDS.
    - The writing lead will assign writing related work to other members while monitoring progress.
- Record keeping: Ellie Kobak
    - The record keeper will ensure that task assignment and completion is being recorded in a complete and timely fashion.
    - The record keeper will summarize any decisions made or issues brought up during group meetings.
    - The record keeper will make sure that any revisions to records or documents are properly noted and archived.
- GUI Lead: Kalyn Koyanagi
    - The GUI lead will monitor progress and verify completion of assigned implementation tasks pertaining to visual and user interface components
    - The GUI lead will routinely check the quality of the GUI code such as performance, style, maintainability etc.
    - The GUI lead will be referred to when any minor design decisions pertaining to visual and user interface components need to be made.
- Backend lead: Kelly Schombert
    - The backend lead will monitor progress and verify completion of assigned implementation tasks.
    - The backend lead will routinely check the quality of the backend code such as performance, style, maintainability etc.
    - The backend lead will be referred to when any minor design decisions pertaining to visual and user interface components need to be made.
- Test lead: Luying Cai
    - The test lead will develop a thorough and robust set of program test cases for the debugging stage. The test lead will assist in developing component test cases where needed.
    - The test lead will check throughout the development process that the currently implemented components are being robustly tested by group members.
    - The test lead will make sure that any issues or bugs that have been raised are corrected in a timely fashion and assign debugging tasks.

The type and number of responsibilities for each member will be flexible to accommodate any issues or imbalances in workload or unforeseen circumstances that may arise. Every member will participate in routine meetings and communicate with each other outside of meetings to make decisions, be assigned tasks, and track the progress as the project goes on.

When a large decision (any decision that may impact other parts of the development process) must be made, the relevant lead will decide on a course of action individually and then propose their verdict to the group. The group will discuss if there are any alternatives that may be preferred or if there are concerns with the decision. If there are no concerns or alternatives, the lead's original decision is carried out. Otherwise, a group decision is made and carried out instead. Smaller decisions that will not impact other pieces to the development process will be left up to the appropriate lead.

2.2. Meetings and Communication

Group members will attend regular formal meetings at the following times:

- Sundays @ 4:30 pm
- Fridays @ 12:00 pm
- *Additional meetings will be scheduled as needed*

Meetings will take place either over Zoom or in the Knight Library depending on the health status of each individual. Each meeting will last 60-80 minutes depending on agenda items and as time permits for individual members.

Group members will attend regular informal check-ins after each CIS 422 class period.

Check-ins will take place in the Price Science Commons with whichever members are available and present for a check-in.

Group members will have discussions and report their progress outside of meetings via the following communication methods:

- Group SMS messaging
- Discord

Members are expected to read such messages to stay up to date on any updates or issues that might arise during development.

3. Work Breakdown Schedule

- Week 1 (1/5 – 1/11)
    - Create build plan
    - Create project plan
    - Create working drafts of the SRS and SDS
    - (Initial project documents due on Tuesday, Jan. 11)
- Week 2 (1/12 – 1/22)
    - Gather sample data
    - Build non-GUI components such as structures for holding student data, randomizing algorithms, and systems of reading to log files or from saved files
    - Build GUI components dealing with visual output and user interface
    - Combine all components to have an executable system ready for testing
- Week 3 (1/23 – 1/29)
    - Begin testing and debugging phase

    - If on or ahead of schedule, consider implementation of 'nice to have' requirements or improvement to required implementation
    - Review and update documentation
- Week 4
    - (Project due on Sunday, Jan. 30)

The responsibility for monitoring progress of each milestone will be in accordance with the roles outlined under the **Management Plan** section. Members will likely have roughly equal distribution of implementation tasks.

4. Monitoring and Reporting

Tasks that have a completion that can be objectively verified will be recorded to a shared spreadsheet. This spreadsheet will also note the group member in charge of completing this task and by what date the task should be completed by. The spreadsheet will be frequently updated to show every self-reported task completion and verification by another group member. Ideally, verification will be done by the relevant lead member or the record keeper.

5. Build Plan

5.1. Plan Details

As further outlined in the SDS, the program will consist of 5 components implemented through 5 files. These files are: main.py, buildQueue.py, fileReader.py, fileWriter.py, and visual.py.

The first files to write are fileReader.py and buildQueue.py. fileReader.py reads an imported tab-delimited file containing the class roster from the instructor and stores any student data necessary for the remaining program execution. buildQueue.py uses the class data obtained by fileReader.py to create a randomized queue of students that determines which students will be "on deck".

After the components concerned with the organization and storing of data are implemented, then work begins on the visual, user interface, and data writing components. These components are written through the visual.py and fileWriter.py files. visual.py handles the display of "on-deck" names as well as the key controls used by the instructor. fileWriter.py responds to key controls from visual.py and writes the relevant student data to log files on the system.

Once each component can execute its individual functionalities and have been tested for appropriate requirements, the final component to link all other components together will be written. This is implemented through the main.py file. This file links the components concerned with receiving and organizing data to the visual and data writing components that need information stored in the backend.

Afterward reaching this buildable state, any remaining ‘must-have’ requirements will be implemented and the possibility of implementing remaining ‘should-have’ or ‘nice-to-have’ requirements will be evaluated.

All resources should be focusing on testing and debugging to reach a stable release candidate by the project deadline unless optional requirements and functionality are being added.

5.2. Rationale

The rationale behind breaking down the system into five components is to separate the most prominent “must-have” requirements into manageable pieces. When the components are separated, the group is also able to ensure that the pieces work individually before attempting to combine the system. This modularity will also give the best environment to attempt incorporating “nice-to-have” requirements.

Development of the backend (non-GUI) and GUI components have been separated to allow group members to work in areas that within their expertise or that they find interesting.

We do expect team members to have more difficulty in creating and integrating the GUI elements as no members have had experience with GUIs. We will have to assess the difficulties that arise with implementing the visual and user interface components early on and reallocate team members accordingly. In addition, even members who are not part of the initial development of GUI components should be required to make themselves familiar with the relevant libraries.

6. Acknowledgements

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

CIS 422 Project 1:
Ducks on Deck
Software Requirement Specification

Kalyn Koyanagi (kek), Ellie Kobak (erk), Kelly Schombert (ks), Liza Richards (ljr), Luying Cai(lc) - 5-6-2019 - v1.0

Table of Contents

1. SRS Revision History

Date	Author	Description
1-8-2022	ljr	Created the initial document and began writing a portion of the first draft to be shared at the second meeting. Revised all "ConOps" and "Specific Requirements" sections.
1-9-2022	ljr	Updated section 2.2, 2.4, 2.6, 3.3, 3.5, and 5
1-9-2022	lc	Updated section 2.3, brief description
1-10-2022	ljr	Updated section 3.2.
1-10-2022	lc	Updated section 2.5.
1-11-2022	ljr	Made final adjustments to sections.

2. The Concept of Operations (ConOps)

2.1. Current System or Situation

The "cold calling" system is designed to assist the instructor in choosing students to answer their questions. The students are randomly chosen in order to encourage participation from everyone in the class, especially those who do not usually raise their hands. This system is meant to evenly distribute the cold calls to ensure that no student is called more or less than another. The system will always be given a list of students in the class to choose from. 4 students will be picked from this given list to be "on deck," which allows those chosen to "warm-up" to being called on. The system is also designed to allow the instructor to "flag" the students who the instructor might want to follow up with after class.

2.2. Justification for a New System

Student engagement is critical to the learning environment. The cold calling system is an easy way to encourage this engagement which therefore promotes students to practice critical thinking skills and learn course material. While "cold calling" is not necessarily a new concept, the opportunity for students to "warm-up" before answering or asking questions is. This could make students more comfortable with cold calling since they have more time to prepare for their turn. In addition, the instructor has the option to flag a student, which makes it easier for the teacher to not only remember to reach out to the student individually but also easier to reach out in general.

With the current system, the instructor has to manually pick and choose students from the roster to call on while keeping track of the number of times each student is called to make sure that no student is called on more than the others. This method is time-consuming, which takes away valuable lecture time. In addition, it is rather distracting to the instructor because they have to continuously switch their focus back and forth from the material they are teaching to filling up the on deck list. This not only makes it hard for the instructor to maintain focus on the lecture, but also hard for the students to focus on the lecture, which can negatively impact their learning. The new cold-calling system takes far less time by doing the picking, choosing, and sharing for the instructor much faster while also allowing the instructor and the students to have an uninterrupted focus on the lecture.

2.3. Operational Features of the Proposed System

The apparent disadvantage of the existing questioning system is that only a few people interact with the professor in each class. Only a small number of people will interact with the professor, and many students may not participate in the classroom. Professors cannot get their feedback, nor can they know whether the students are understanding or focusing on the content of the class. Therefore as a solution, the cold call system can help the professor solve the problem of being unable to interact with students who are afraid of raising their hands. The selection of students in the cold call system is random, so there will be an equal distribution of participation opportunities. After getting the student's response, the professor can choose to mark the student to facilitate follow-up on the student's questions or topics of interest after class.

2.4. User Classes

The main user class for this system is the instructor. The instructor will use the cold-calling system to randomly select students to answer their proposed questions or ask questions. The motive for the instructor to use this system during class is to encourage participation during lectures which in turn helps students learn lecture material and practice critical thinking skills. This system makes it easier for the instructor because they do not have to manually take valuable lecture time to continuously pick out random students to cold-call on. This system will also help the instructor keep track of the participation of each student in the class and reach out to them if necessary.

2.5. Modes of Operation

The mode instructors can use the cold-call system to randomly select a student to answer the question during the lecture. Also, the instructor can use Arrow keys to flag students whom they want to follow up with after class. All these processes go through the instructor's laptop, and students cannot see anything during the lecture. On the other hand, students can gain more and better experience during class by this cold-call system since this system randomly selects students from the list, they will focus more rather than only a few students participating in class.

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

Use Case: Day-to-day usage

Brief Description: 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.

Actors: A student and instructor

Preconditions: An instructor wants to encourage students to participate in class discussions, is comfortable "cold-calling" on students, wants to make sure each student is treated equally with the cold-calling, and uses a computer as part of the in-class presentation.

Steps to Complete the Task:

1. The instructor is in the classroom with their computer, ready to start the lecture. They plug in their computer to the projector and open Powerpoint and start lecturing.
2. While lecturing, the instructor wants to check in with the class to see if students are comprehending lecture material, so they begin asking questions to the class and wait for a response.
3. When no students offer to respond to the question, the teacher uses the cold-call-assist software installed on their laptop. The instructor searches in the hard drive for the program within 5 seconds.
4. When the system is opened, a list of four student names automatically appears in an "on deck" horizontal window positioned to take up as little space as possible to avoid blocking the presentation. The instructor places the focus back on the Powerpoint slides, but the on deck window stays in the front.
5. The instructor can note which students were cold-called at the beginning or end of the previous class to ensure that the system is randomized between run times.
6. The instructor asks if the students on the list would like to answer the question.

Postconditions: The system stops running, but is still ready for the next time it is used.

Use Case: After class, the instructor reviews if any students might benefit from encouragement.

Brief Description: 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.

Actors: The instructor

Preconditions: After class, the instructor knows that they did some cold-calling in class that day and that a student demonstrated special interest in a topic. They want to email a conference paper on the topic to that student.

Steps to Complete the Task:

1. Teach the class using cold calling software
2. After class, the instructor reviews the daily log file and its contents. At the top of the file, there is a heading to indicate that this is the daily log file for the cold call-assist program.
3. Today's date is under the heading, and then there is one line for each cold-call that is made that day. Each line is formatted as:

   <response_code> <tab> <first name> <last name> "<" <email address> ">"

   The response code is blank if there was no flag and "X" if there was.
4. The instructor can recall how the student acted and participated in class. One student was having too many side conversations and did not hear when they were called on, so the instructor wants to email this student to remind them to pay more attention in class. Or the instructor can remember that the student has a special interest in a topic and wants to send them a paper or article regarding the topic.
5. The instructor drags the mouse across the portion of the line containing the students' email addresses in brackets and uses the copy command. Then paste this email address into the "To:" header of her email client.
6. The instructor composes and sends the email.

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

Use Case: At the end of the term, the instructor reviews a summary of class participation.

Brief Description: The instructor will use students' term performance data to see their cold-call performance. Data import into spreadsheet will using <total times called> and <number of flags> to compute a cold-call participation score for students. Those scores can be imported into Canvas for students to check.

Actors: The instructor

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.

Steps to Complete the Task:

1. The instructor opens the summary performance file and reviews its contents
2. The instructor will see the heading at the top of the file, which indicates this is the summary performance file.
3. The instructor sees the heading of the file, which indicates this is the summary performance file for the cold-call-assist program, and headings for the columns in the lab-delimited file.
4. The instructor sees the list of students with performance data after each student, each line will be formatted as:

   <total times called> <number of flags> <first name> <last name> <UO ID> <email address> <phonetic spelling> <reveal code> <list of dates>

   There should be 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, and they are formatted as YY/MM/DD and are in chronological order.

5. 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.
6. 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 for their preparation for being called-on in the class

3. Specific Requirements

3.1. External Interfaces (Inputs and Outputs)

Must-Have:

1. Arrow Keys

   Description: Used to choose and move between names in the "on deck" list for them to be removed and potentially flagged.

   Source of Input: User input from pushing the arrow keys on a keyboard.

   Valid ranges of inputs and outputs: Up, down, left, and right arrow keys

   Units of measure: None

   Data formats: None

2. Student List

   Description: A file containing each student's first name, last name, UO ID, email address, phonetic spelling, and reveal code that is tab-delimited and is uploaded into the system.

   Source of Input: User input

   Valid ranges of inputs and outputs: .txt file

   Units of measure: Megabytes (MB)

   Data formats: Tab delimited and formatted as <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 is used as a place for the instructor to write notes.The first line until the first <LF> will be a comment that is to not be modified.

3. End of Term Log of Cold Calling Information

   Description: A file containing each student's first name, last name, UO ID, email address, phonetic spelling, and reveal code that is tab-delimited and is to be uploaded into the system. This file may already contain past cold calling data.

   Source of output: .txt file

Valid ranges of inputs and outputs: .txt file

Units of measure: Megabytes (MB)

Data formats: Tab-delimited and formatted as <total times called> <number of flags> <first_name> <last_name> <UO ID> <tab> <email_address> <phonetic_spelling> <reveal_code>. The UO ID will be nine digits. The reveal code is used as a place for the instructor to write notes.The first line until the first <LF> will be a comment that is to not be modified.

4. Cold Calling Display

   Description: A small, horizontal window at the top of the user's screen above lecture materials with four student names. It will sit in the foreground of other open applications and will listen to keystrokes while being in the background.

   Source of output: The user's screen

   Valid ranges of inputs and outputs: A small, horizontal window displaying four names.

   Units of measure: None

   Data formats: Small, horizontal window at the top of the screen.

3.2. Functions

Must-Have:

1. Arrow Keys

   Validity checks on the inputs: Ensure that each arrow correctly corresponds with its associated movements.

   Sequence of operations in processing inputs:
   1. Check which arrow key was pressed first
   2. If the left arrow key is pressed first, the name at position one is highlighted first. If the right arrow key is pressed first, the name at position four is highlighted first.
   3. From the starting positions, use the left and right arrow keys to move between students whose names should highlight as they are reached.
   4. Once the desired name is reached, either use the up arrow key to flag and remove the student from the current on deck queue or use the down arrow key to only remove the student from the queue.
   5. After a student is removed from the queue, add another random student to the on-deck list visible on the screen.
   6. If a student is not flagged using the up arrow, then an "X" will replace the <response_code>. If the student is flagged, then the instructor will type out any notes or comments he has on the student's response or question.

   Responses to abnormal situations:
   1. With the initial opening of the program and no names are highlighted, if the left arrow key is pressed, then the name in the first position is highlighted; if the right arrow key is pressed, then the name in the fourth position is highlighted.
   2. If you are in an end position like one or four and you press the left or right arrows, then no movement happens.

Relationship of outputs to inputs:

a.) input/output sequences: Input is the user input from keystrokes. Output is the associated event that occurs from pressing an arrow key.

b.) formulas for input to output conversion: Taking user input of the keystroke and performing the correct movement or action associated with the arrow keys is the output.

2. Student List

Validity checks on the inputs: Making sure that the file is of the correct type and correct format. Ensure that Each information item for each student is of the correct format.

Sequence of operations in processing inputs:

1. At the start of the program, the system should import a file.
2. Upon import of the file, it will be checked if it is of the correct type.
3. Also upon import of the file, it will be checked if it is of the correct format.
4. If it is of the correct type and format, the system will continue with the import of the file for its use in the system.

Responses to abnormal situations:

1. If the file is not of the correct type, display an error indicating that the file is not of the correct type, and the file should not import until it is the correct type.
2. If the file is not of the correct format, display error indicating that the file is not of the correct format and the file should not import until it is the correct type.
3. If no file is provided, display an error message indicating that a file of a certain type and format is necessary to continue.

Relationship of outputs to inputs:

a.) input/output sequences: File is inputted into the system, and the output is using the file contents in the display. Output also includes daily logs and end of semester summary containing collected cold-calling data

b.) formulas for input to output conversion: Once the file is imported, it's contents will be parsed into a queue to be used by the system. While the system is running, this file will be collecting the cold-call data and storing it in a tab-delimited file that can be exported and seen by the user.

3. End of Term Log with Cold Calling Information

Validity checks on the inputs: Making sure that the file is of the correct type and correct format. Ensure that Each information item for each student is of the correct format.

Sequence of operations in processing inputs:

1. At the start of the program, the system should import a file.
2. Upon import of the file, it will be checked if it is of the correct type.
3. Also upon import of the file, it will be checked if it is of the correct format.
4. If it is of the correct type and format, the system will continue with the import of the file for its use in the system.

Responses to abnormal situations:

1. If the file is not of the correct type, display an error indicating that the file is not of the correct type, and the file should not import until it is the correct type.
2. If the file is not of the correct format, display an error indicating that the file is not of the correct format and the file should not import until it is the correct type.

Relationship of outputs to inputs:

a.) input/output sequences: File is inputted into the system, and the output is using the file contents in the display. Output also includes daily logs and end of semester summary containing collected cold-calling data

b.) formulas for input to output conversion: Once the file is imported, its contents will be parsed into a queue to be used by the system. While the system is running, this file will be collecting the cold-call data and storing it in a tab-delimited file that can be exported and seen by the user.

4. Cold Calling Display

Validity checks on the inputs: Make sure that the display is in the correct position of the screen window. Make sure it is of the correct size. Make sure that it responds to keystrokes while in the background.

Sequence of operations in processing inputs:

1. If an arrow key is pressed, the display should update and respond to this keystroke and move to highlight the next name.
2. If an arrow key is pressed while the system is in the background of other apps, the system should still update and respond to the keystroke and move to highlight the next name.

Responses to abnormal situations:

1. If the display is not in the correct position on the screen
2. If the display is too small or too large
3. If the display does not respond while in the background of other apps

Relationship of outputs to inputs:

a.) input/output sequences: The input into the display will be a tab-delimited file containing student information. The output of the display will be the window. The input into the display will also be the arrow keys, and the output

b.) formulas for input to output conversion: Display always responsive to user input from keystrokes. Uses keystrokes to add information to output files.

3.3. Usability Requirements

<u>Required:</u>

Must-Have:

1. A file containing every student in the class and their associated information.
2. A display with four random student names visible to be chosen from using the arrow keys
3. Arrow keys the user can use to choose and move between students.
4. A file containing all cold-calling data collected from previous uses, which can be imported and exported.
5. Keystrokes must work while the application is in the background.
6. A flagging system for the instructor to write notes on student responses.

Should Have:

7. The system should be up and running after a second of double-clicking on the icon.
8. A daily log is provided after system usage that shows the cold calling data from that day.
9. The student names should shift in the on-deck list after a name is removed.

10. End of term summary of each student's performance across all of the times that each student was cold-called during the term.
11. The system should run with any number of students on the student list.
12. On deck display must take up as little screen space as possible.
13. The system must run and respond to keystrokes while in the background of other applications.
14. Save cold-calling data to a file after every keystroke so that no data is lost when exiting the system.
15. At system startup, the random number generator will be re-seeded with a different seed, so the system is ensured to be random.

Not Required:

Could Have:

16. An optional secondary system which is a photo-based name learning system.
17. The system should not generate any error messages that interfere with the running of the program.
18. The instructor chooses a number of students to be on the on-deck list.
19. Is possible to change the system's use of tab-delimited files to the use of comma-delimited files.
20. A way to switch all key mappings.

3.4. Performance Requirements

Static:

1. .txt file of the class roster with any amount of students.
2. .txt file containing cold calling data.
3. The software runs and responds to keystrokes while in the background of other applications.
4. Keystrokes from user input.

Dynamic:

1. System startup within one second of double-clicking on the application
2. When the system starts, the random number generator will be re-seeded.
3. No data is lost during runtime or upon exit.

3.5. Software System Attributes

Software Attributes:

1. Reliability
2. Security
3. Privacy
4. Maintainability
5. Portability

Most Important Attributes:

1. System must run on Macintosh OSX 10.13 (High Sierra) or 10.14 (Mojave).
2. All system-related and system-development-related documents that are intended for human reading bust be in either plain text or PDF.
3. System will be built using Python 3 along with The Python Standard Library, but no other imports except for pyHook.
4. Python code must run in Python 3.7 through 3.10.
5. Instructions must be provided for how to compile the code.
6. No server connections may be required for either installing or running the software.
7. No virtual environments may be used.
8. No gaming engines such as Unity may be used.
9. There can be at most 20 user actions to compile the code and run the program.
10. An experienced computer programmer should not require more than 30 minutes working alone with the submitted materials to compile and run the code.

4. References

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

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

Tom, Sherrington. (2021). "Cold Calling: The #1 strategy for inclusive classrooms - remote and in person". Available at: https://teacherhead.com/2021/02/07/cold-calling-the-1-strategy-for-inclusive-classrooms-remote-and-in-person/

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

5. 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.

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

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.

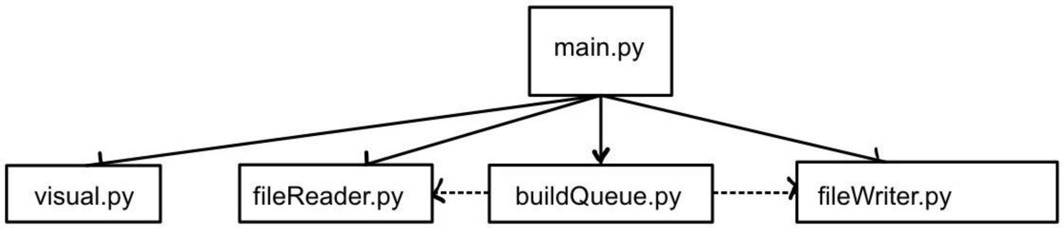


Figure 1. Components of Ducks on Deck

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

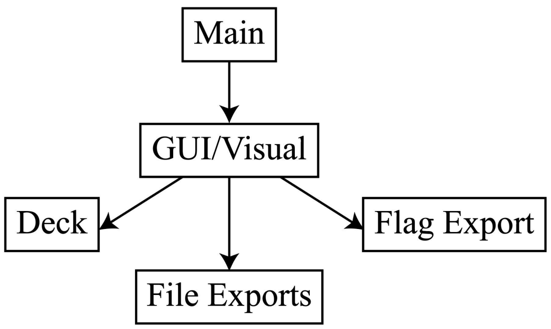


Figure 4.1.1 Static Model of Instructor Interface

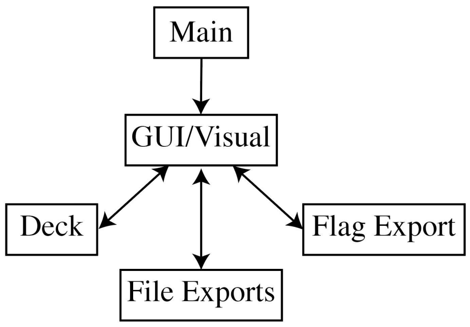


Figure 4.1.2 Dynamic Model of Instructor Interface

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

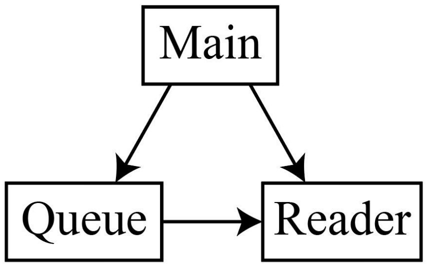


Figure 4.2.1 Static Model of Queue Interface

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

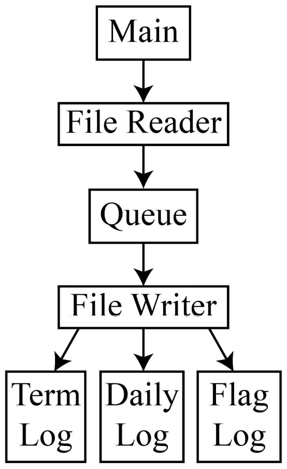
The module's role and primary function.

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



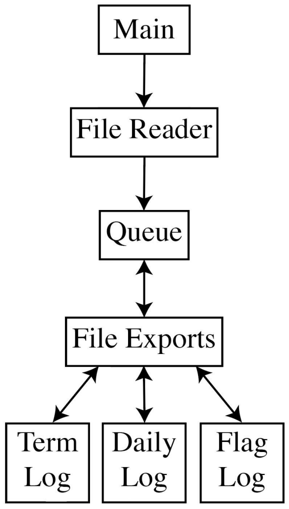


Figure 4.3.1 Static File Usage Model

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

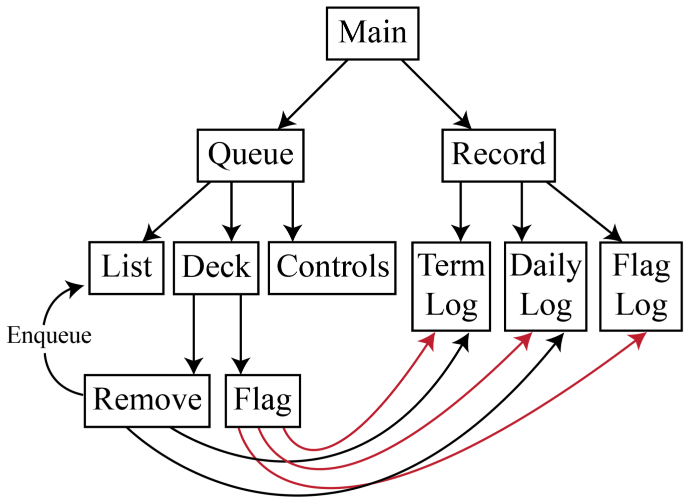


Figure 4.4.1 Dynamic Model of Record System Interface

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.

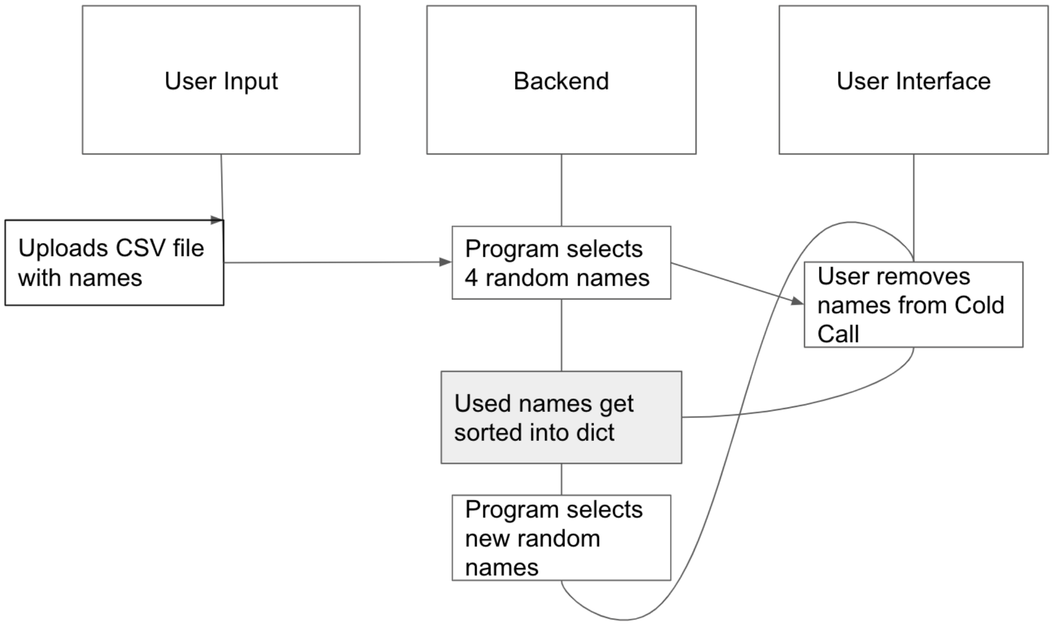

User Input
Backend
User Interface
Uploads CSV file with names
Program selects 4 random names
User removes names from Cold Call
Used names get sorted into dict
Program selects new random names

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.

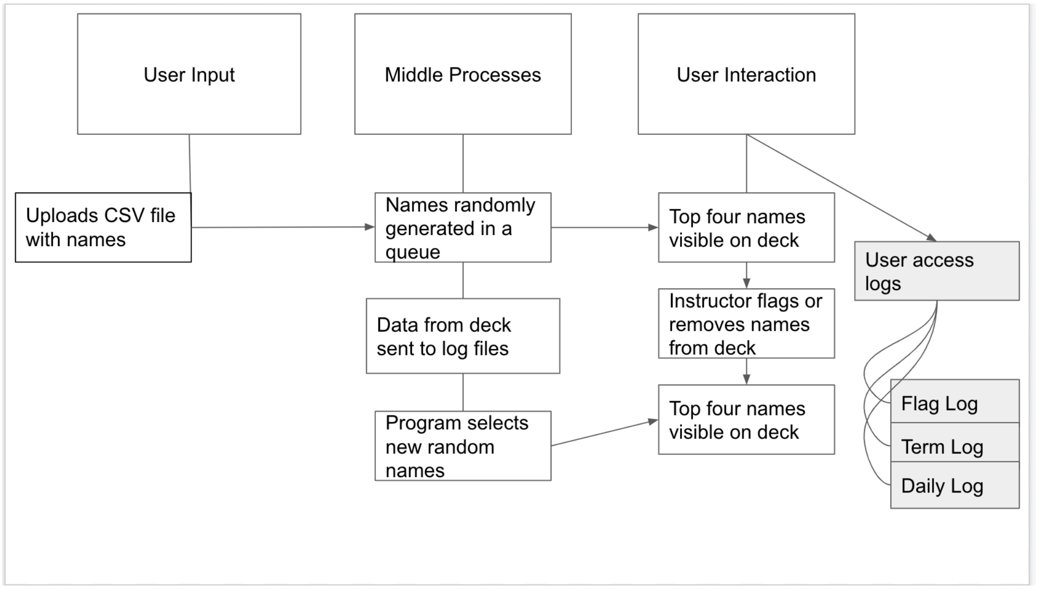

User Input
Middle Processes
User Interaction
Uploads CSV file with names
Names randomly generated in a queue
Top four names visible on deck
User access logs
Data from deck sent to log files
Instructor flags or removes names from deck
Program selects new random names
Top four names visible on deck
Flag Log
Term Log
Daily Log

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.