CIS 422 Project 1: Classroom Cold-Call Assist Software Software Design Specification

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

January 17th, 2020 - v1.0

Table of Contents

1. SDS Revision History

Date	Author	Description
1-14-2020	mc	Created the initial document.
1-14-2020	mc	Started writing sections for v0.1
1-16-2020	op	Finished section 4
1-16-2020	mc/op	Finished SDS version v0.1
1-17-2020	jg	Formatting updates and minor edits < ***first working draft***

2. System Overview

2.1. Description of System

This system will provide an instructor with the basic capabilities of "cold calling." "Cold calling" is a method in which an instructor calls on a student who hasn't recently participated in the class's discussion. This software will also provide the students with a "warm-up period." This is established through a queue that displays the top 4 students that may be "cold called" next. The instructor will also have the capability to remove students from the queue when they have participated in class discussion and flag students they want to contact after class. The main goal of this system is to "cold call" on students efficiently in order to improve class discussion and overall participation. This software also has the capability of aiding an instructor with learning their student's names through flashcard mode that is styled much like Quizlet. Each flashcard has the student's picture on one side and their name on the other side.

2.2. System Organization and Interaction

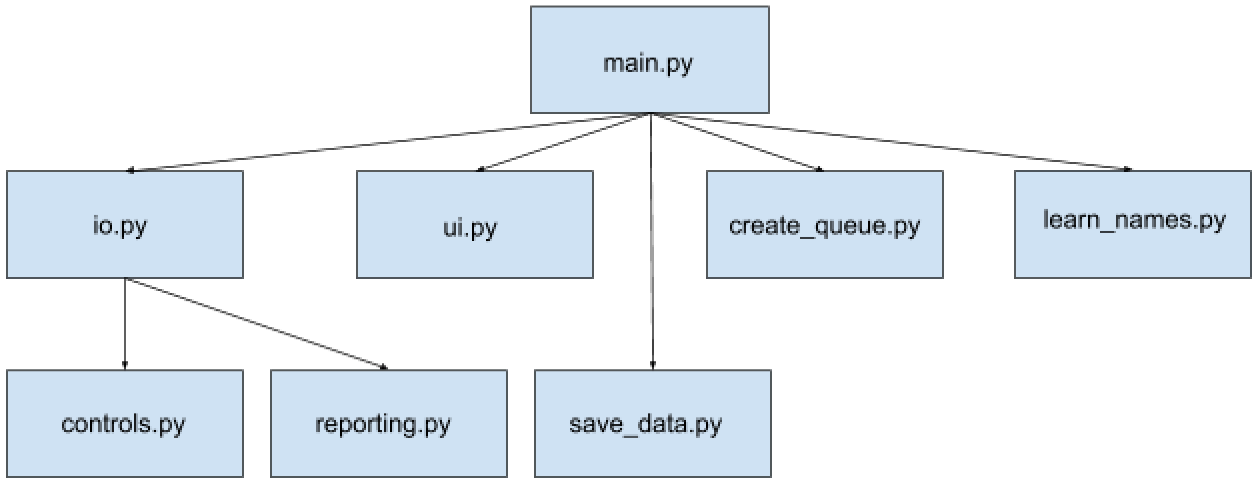
The system is divided into three functionalities: "cold calling," name flashcards, and emailing students. These three functionalities are broken up into a total of 8 files.

1. ***main.py*****:** ties the program together and is the main program file that runs the overall program.
2. ***io.py*****:** reads and parses the class text files.
3. ***ui.py*****:** contains the user interface graphics.
4. ***create_queue.py*****:** contains the algorithm that sorts the students in a class into a queue that is used to cold call the students.
5. ***save_data.py*****:** saves the state of the queue and its data when the program is either paused or exited.
6. ***reporting.py*****:** gives the professor the ability to keep track of students they want to email and allow them to email them.
7. ***controls.py*****:** holds and creates all the controls of the program. Such as removing a student from the queue after they have spoken and flagging a student to email later.

8. ***learn_names.py*:** holds contents necessary for the user to learn the students names much like the format of Quizlet.

3. Software Architecture

The system is divided into three functionalities: "cold calling," name flashcards, and emailing students. These three functionalities are broken up into a total of 8 files.



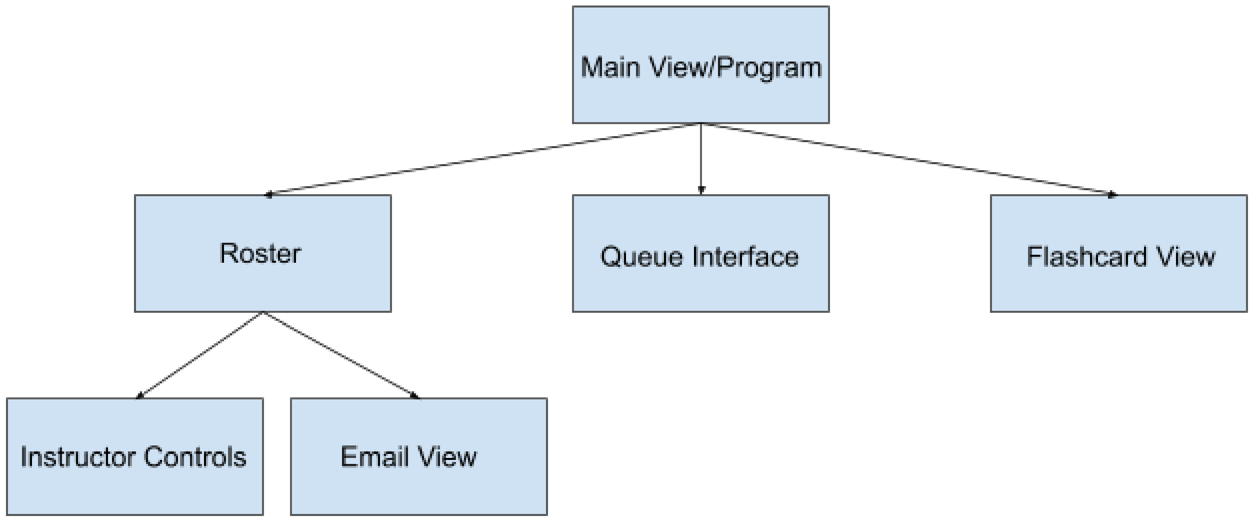
(Figure 3.1) The files that make up cold calling, name flashcards, and emailing students.

1. First file (main.py): ties the program together and is the main file that runs the overall program.
2. Second file (io.py): reads and parses the class text files.
3. Third file (ui.py): contains the user interface graphics.
4. Fourth file (create_queue.py): contains the algorithm that sorts the students in a class into a queue that is used to cold call the students.
5. Fifth file (save_data.py): saves the state of the queue and its data when the program is either paused or exited.
6. Sixth file (reporting.py): gives the professor the ability to keep track of students they want to email and allow them to email them.
7. Seventh file (learn_names.py): holds contents necessary for the user to learn the students names much like the format of Quizlet.
8. Eighth file (controls.py): holds and creates all the controls of the program. Such as removing a student from the queue after they have spoken and flagging a student to email later.

These eight files we split into four steps. During the first step the files io.py, ui.py, and create_queue.py will be created. During the second step we the files save_data.py and controls.py will be created. During the third step the main.py will be created. Lastly, during the fourth step the files reporting.py and learn_names.py will be created.

The system is split into these steps in order to account for functionality, ease of testing, and priority. The first step implements the basic individual functions of the program. These are established first so that the smaller features can be tested individually. The second step implements the more advanced attributes that build off of the first step and sets up the user controls for the program. This allows the control of the smaller functions to be tested before the program is entirely linked. Finally, the third step connects the individual functionalities into one connected program. This aspect was saved to be implemented closer to the end of the project creation in order to first allow the operation of more minor components. The final step saves the least important functionalities for last in case there isn't enough time to implement each aspect of the program.

These files have been condensed into the following modules:



(Figure 3.2) A condensed diagram of files from Figure 3.1.

The main view/program is the module where the main program is stored and ran. The roster module reads in the text files of students and translates them for the programs "cold call" queue. The instructor controls module contains the controls the user has when operating the program. The email view module allows the user to flag a student and email those students later on. The queue interface module establishes the "cold call" queue and the order in which students will be called on. Finally, the flashcard view module allows the user to learn the students names through a flashcard based environment modeled from Quizlet's template.

4. Software Modules

4.1. Main View/Program Module

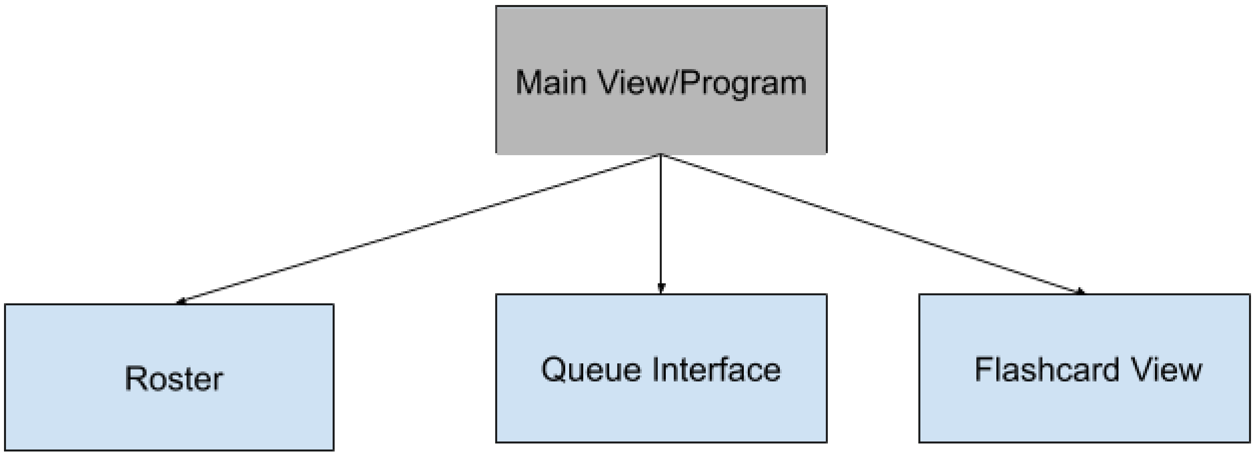
The module's role and primary function.

The main view or program module's primary purpose is to run the main program and tie the individual functionalities into a single program. This will be achieved by calling the other modules in order to have a cohesive cold calling system. This module contains the main program, the saving program, and the user interface.

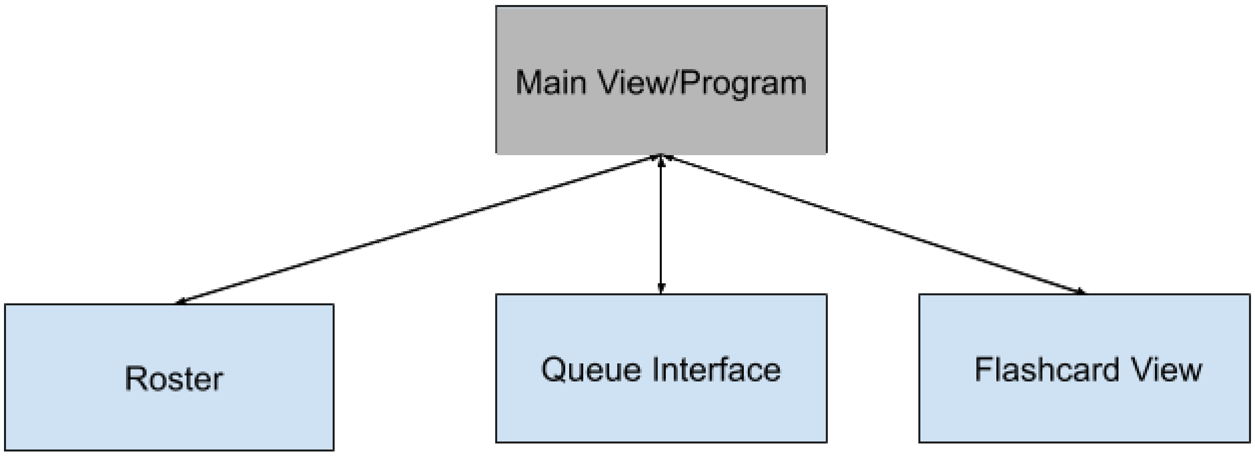
Interface Specification

This program will interact with the following modules: Roster, Queue Interface, and Flashcard View. It will call different programs within these modules depending on the intended service of the user. For example, if the user wants to learn the names of the students the main view module will interact with the flashcard view module.

A Static and Dynamic Model



(Figure 4.1.1) Main View/Program Module Static Model



(Figure 4.1.2) Main View/Program Module Dynamic Model

Design Rationale

This module is designed to interact with every other module either directly or indirectly. Since it is the main view, it is made to bring every subsystem together in order to create a working system. The main view specifically includes the main program, save queue feature and the user interface. These three features were decided to be included within the subsystem main view because they are the important ground layer of the whole working system.

4.2. Roster Module

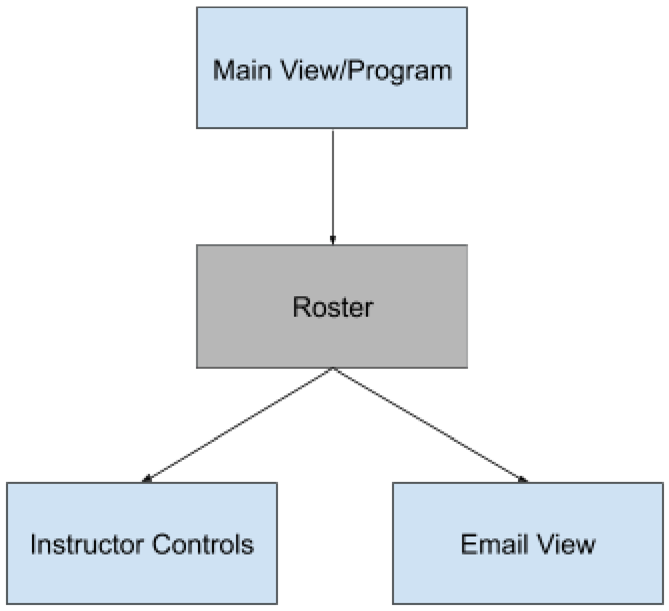
The module's role and primary function.

This is the main module that will deal with the input and output of the program. One of the main goals this module has is to read and parse the input text files for a given class. These files will contain a list of students and their student ID numbers. These files will have their information separated by tabs. Each line will then be read and processed as a single student that will be added to the queue. This module will also handle reading of the .jpg files that will be used in the flashcard mode in the flashcard view module.

Interface Specification

This module will interact with the Instructor Controls module, Email View module, and the Main View/Program module. Roster is used to communicate the student list to the email view and the instructors control. It is used from main so that main can take the student list and use it to interact with other modules.

A Static and Dynamic Model



(Figure 4.2.1) Roster Module Static Model

Design Rationale

Initially this module did not exist and its components were combined into other modules. However, due to the importance of input and output we thought it should be created into its own module. It was decided to communicate with two subsystems underneath it, instructor control and email view. It also passes its information to the main which will call for the queue to be created. It is important for roster to become into its own module so that the main view will not become too cluttered and confusing with concepts.

4.3. Instructor Controls Module

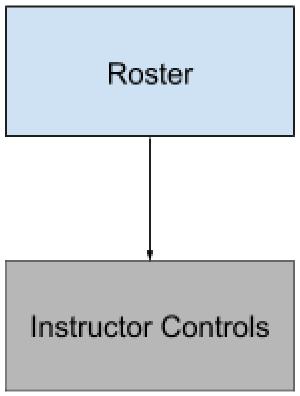
The module's role and primary function.

This module is in charge of handling the program's user controls. It is responsible for allowing the user to navigate around the system, manage the queue, flag students, and email the flagged students. It is in charge of sending signals between the other modules and giving the user complete control of the program's capabilities.

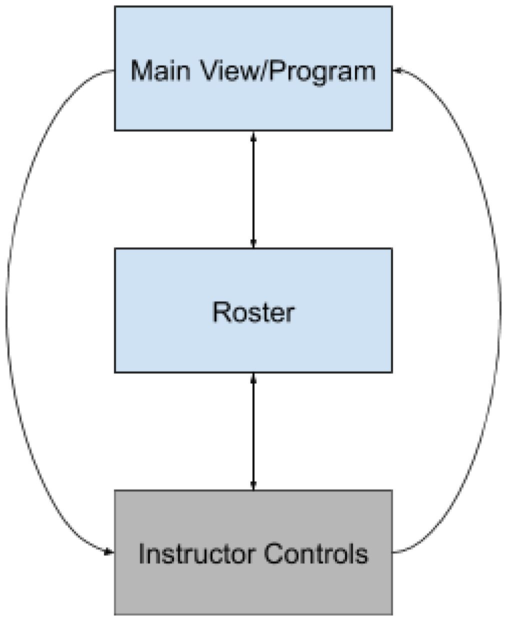
Interface Specification

This module will interact with the Roster module. Instructor controls module allows signals to be exchanged in order to manage the queue, flag students, and email students using the Roster module.

A Static and Dynamic Mode



(Figure 4.3.2)
Instructor Controls Module Dynamic Model



(Figure 4.3.1)
Instructor Controls Module Static Model

Design Rationale

This module is designed to control the program. It was created to separate the base of the program from the commands that would navigate the user around the system. This is necessary because it is important to split functionality and use. This module explicitly will define how the user removes a student from the queue, flags a student, starts the "cold calling" program, ends the "cold calling" program, and begins a classes name flashcards.

4.4. Email View Module

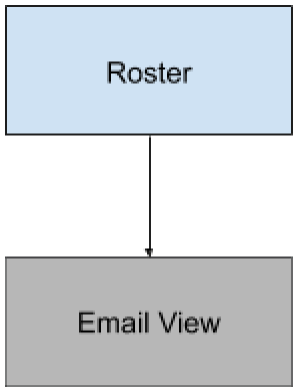
The module's role and primary function.

This module's main role is to add the capability of flagging students, collecting data on the flagged students, and emailing these students.

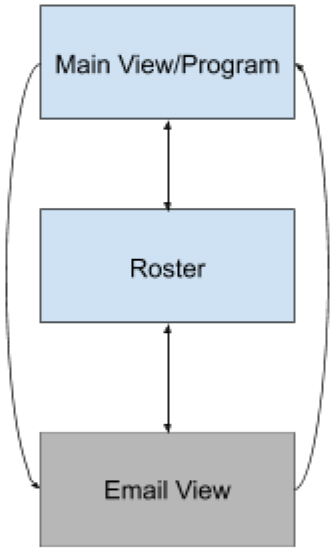
Interface Specification

This module will interact with the Roster module. The Roster module will receive the controls on how to flag a student and the data needed for the user to email the flagged student.

A Static and Dynamic Model



(Figure 4.4.1)
Email View Module Static Model



(Figure 4.4.2)
Email View Module Dynamic Model

Design Rationale

This module is designed to control the functionalities behind the emailing feature of this program. It was created to separate the email feature from the rest of the program. This design decision was made because the ability to email a student through this program is low priority. The idea was that if time did not permit the implementation of this feature, then it would not affect the rest of the program.

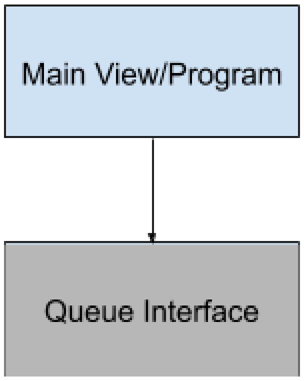
4.5. Queue Interface Module

The queue interface module's main purpose is to establish the "cold call" line and the order in which students will be called on. The list of students will be held in a queue which will dequeue and enqueue students to be in line for the cold call.

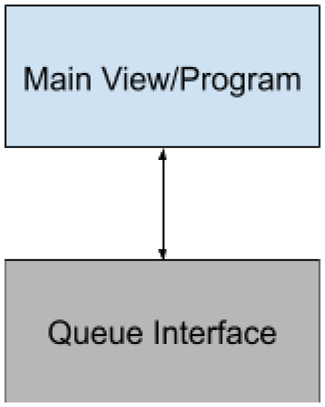
Interface Specification

This module will interact with the Main View/Program module. The Main View/Program module will call the Queue Interface module to access functions related to the "cold calling" aspect of the system. The Queue Interface module will then offer an order of students and save the programs data to the Main View/Program module.

A Static and Dynamic Model



(Figure 4.5.1)
Queue Interface Module Static Model



(Figure 4.5.2)
Queue Interface Module Dynamic Model

Design Rationale

The Queue interface module is directly underneath the main because it updates the queue which must be saved within the Main View. Initially the queue interface was considered to interact with many subsystems but instead the main view will interact with every other subsystem for the queue interface.

4.6. Flashcard View Module

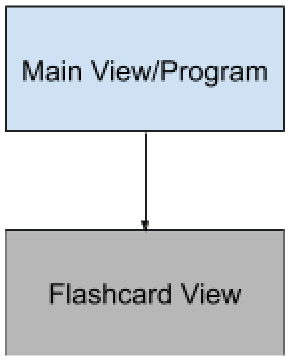
The module's role and primary function.

This module's main role and primary function is to allow the user to learn the students names through a flashcard based environment modeled from Quizlet's template.

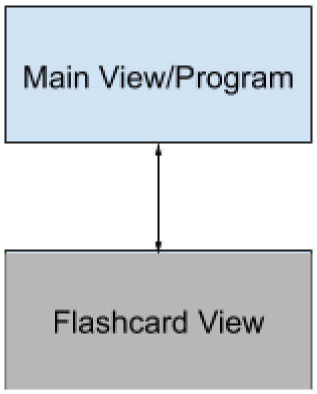
Interface Specification

This module will interact with the Main View/Program module. The Main View/Program module will call the Flashcard View module to access functions related to the learning names aspect of the system.

A Static and Dynamic Model



(Figure 4.6.1)
Flashcard View Module Static Model

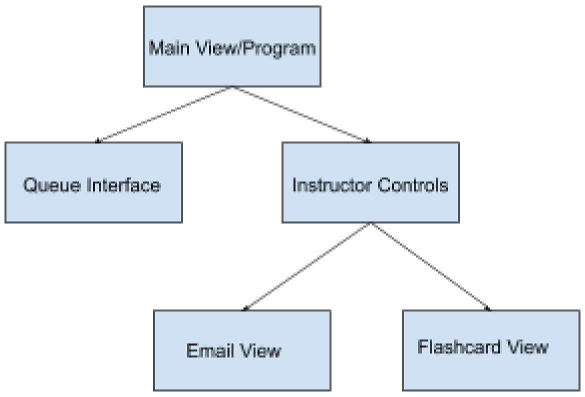


(Figure 4.6.2)
Flashcard View Module Dynamic Model

Design Rationale

This module is designed to control the functionalities behind the flashcards feature of this program. It was created to separate the flashcards from the rest of the program. This design decision was made because the ability to learn students names is low priority. The idea was that if time did not permit the implementation of this feature, then it would not affect the rest of the program.

4.7. Alternative Designs



(Figure 4.7.1) Alternative/Initial design of subsystems.

Figure 4.7.1 is the initial design that was considered for the architecture of this system, however we decided against this model due to a need for a new subsystem, Roster, that deals with the input and output of the program. As well as moving subsystems that interact with this new model into new places seen in Figure 3.2.

5. Dynamic Models of Operational Scenarios (Use Cases)

There are three uses for this system. These uses include "cold calling" as defined previously, learning the names of an instructor's students, and emailing students to the system.

5.1. "Cold Calling" Use Case

This use case is used to "cold call" on students

1. User opens application
2. User interface is displayed
3. User selects class to "cold call"
4. System loads the class's data and the class's past queue information
5. User starts the queue
6. The "cold calling" feature is now active
7. Queue is displayed
8. User removes and flags students
9. User ends system
10. Queue data is saved
11. Email data is created
12. User can now restart queue, close application, learn student's names, or email students

5.2. Learning Names Use Case

This use case is used to learn the names of an instructor's students

1. User opens application
2. User interface is displayed
3. User selects class
4. System loads the class's data and the class's past queue information
5. User opens students flashcards
6. System loads class's flashcards
7. User flips, skips, and removes and flashcards from deck
8. User ends system
9. User can now restart queue, close application, learn student's names, or email students

5.3. Emailing Students Use Case

This use case is used to email students

1. User opens application
2. User interface is displayed

3. User selects class
4. System loads the class's data and the class's past queue information
5. User opens email feature
6. System loads flagged students
7. Flagged students are displayed
8. User emails students
9. User ends system
10. Email data is updated
11. User can now restart queue, close application, learn student's names, or email students

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.

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

IEEE Std 1016-2009. (2009). IEEE Standard for Information Technology—Systems Design—Software Design Descriptions. https://ieeexplore.ieee.org/document/5167255

Parnas, D. L. (1972). On the criteria to be used in decomposing systems into modules. *Commun. ACM*, 15(12), 1053-1058.

7. Acknowledgments

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 1016-2009.

The project creation process is based off of ideas found in the textbook: Hans van Vliet. (2008). *Software Engineering: Principles and Practic*e, 3rd edition, John Wiley & Sons.