Software Requirements Specification

for

Voting System

Version 1.0

Prepared by Jacob Malin, Romeo Sweeney,

Nathan Kjaer, Shruti Jayavardhanan

University of Minnesota

February 2024

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Voting System	2/2/2024	Creation of document and writing the purpose of document	0.1
Voting System	2/9/2024	First Draft, filling out most sections	0.2
Voting System	2/11/2024	Final Draft, major revisions and filling out function requirements	1.0

1. Introduction

1.1 Purpose

The purpose of this document is to present a detailed description of a voting system that uses OPL or CPL to count and compute ballots. The voting system does not include the process of gathering ballots or adding candidates; it is an original system that interfaces with a ballot collection system through an input file. This document will explain the purpose and features of the system, what the system will do, and the constraints that it will run under for version 1.0. This document is intended to be used by users of the system as well as developers of the system.

1.2 Document Conventions

This document was written based on the example IEEE template for System Requirement Specification Documents.

1.3 Intended Audience and Reading Suggestions

- Typical users, such as election officials and administrators, who want to use this software to calculate election results accurately.
    - Typical users would be interested in the section Product Functions, which details the high-level function of the system.
    - Typical users would also be interested in the sections User Interfaces and Text Prompt File Input, which describe the user interface.
- Programmers who are developing the system and would like more clarification on the system requirements, are fixing bugs within the current system, or are developing the system further.
    - Programmers should start with the section System Features, which describes the function requirements for the system, and then should refer to the rest of the document for further clarification.
    - Programmers should refer to the section Software Interfaces for the input file constraints.
- Testers who would like to know what they must test for, and the constraints on the system.
    - Testers will be interested in the section Other Nonfunctional Requirements, which details many non-functional requirements of the system.

1.4 Product Scope

This is a tool for voting officials to compute the results of an election according to the rules of OPL and CPL. This software will allow election officials to calculate elections faster, more efficiently, and accurately than other systems.

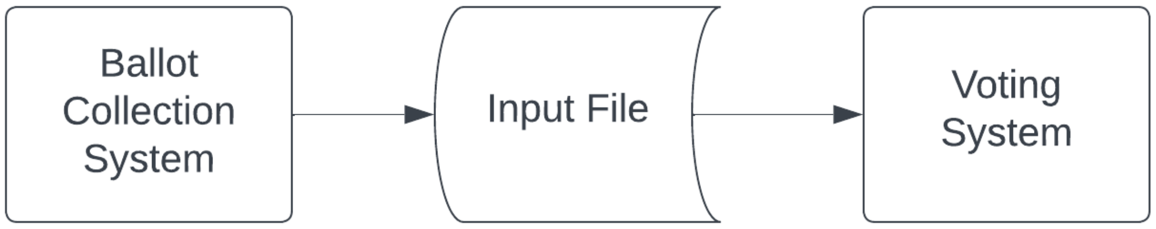
1.5 References

- IEEE Template for System Requirement Specification Documents, provided by Professor Shana Watters on Canvas.
- Use Case Template, provided by Professor Shana Watters on Canvas.
- Voting System Requirements Document, provided by Professor Shana Watters for Spring 2024 on Canvas.
- There is a separate Use Cases Document, named UseCases_Team3.pdf that is within the SRS folder of the GitHub Repository for this project.

2. Overall Description

2.1 Product Perspective

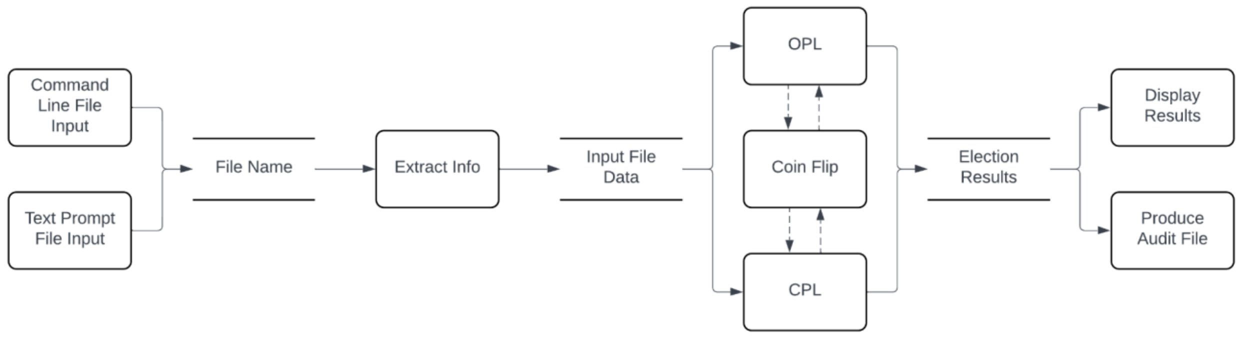
This is a new piece of software that integrates into an existing ballot collection system by an input file. This ballot collection system is unaffiliated with our product. The input file contains information about the type of election, number of seats, number of ballots, information about parties and candidates, and the ballots. The format of the input file is set and may not be modified.



2.2 Product Functions

- To start the program:
    - In a command line environment, the program can be run with a file name as input.
    - The program can be run by double-clicking on the executable file or running the program with no command line arguments to bring up a text prompt.
        - Text prompt asks for the file name, and the user inputs the file name and submits it with enter.
- Once the file name is inputted, the voting system opens the file input file for reading.
- The system parses the entire file and extracts data to store a data structure.
- Depending on the voting system specified in the input file, the system will run the CPL or OPL algorithm.
    - If the first line says CPL, the system runs the CPL algorithm.
    - If the first line says OPL, the system runs the OPL algorithm.
- The system will assign seats based on the LRASA.
    - See the Main Course of use case UC_CPL or UC_OPL in the Use Cases Document for a description of the LRASA.

- In the case of a tie, the system will select a winner based on a random number generator that is run 1000 times per candidate to ensure fairness.
- Once the calculation is finished, the system will save the results to a file.
- The system will also display the results on the screen.
- The user may then close the program.



* Dashed lines within the graph represent optional paths.

2.3 User Classes and Characteristics

- Election Officials: They will run this system during an election. Election officials are our most important user class because they are the intended end users. Election officials use the text prompt.
- Testers: Test the system for different inputs, unit testing, integration testing, and stress testing. Testers use the command line input.
- Programmers: The architects of the program will need access to it to provide implementation, maintenance, and enhancements. Programmers use the command line input.

2.4 Operating Environment

The operating environment is Linux, specifically the University of Minnesota, Computer Science & Engineering lab machines.

2.5 Design and Implementation Constraint

- The voting system is implemented strictly in C++.
- It uses C++'s Standard Template Library (STL) for imported functions and data structures.
- The system will have a makefile for compilation instructions.
- The program must process 100,000 ballots in under 4 minutes.
- There are no expectations for system maintenance.

2.6 User Documentation

There are no intentions to create user documentation as of right now.

2.7 Assumptions and Dependencies

The Voting System is developed in C++ and therefore requires C++ to be installed on the user's system. The latest version of this Voting System requires version 11.4.0 in a CSE lab machine environment.

3. External Interface Requirements

3.1 User Interfaces

There are three user interfaces, the program start-up and file prompt associated with the requirement Text Prompt File Input, the final display of results associated with the requirement Display Results, and the audit file associated with Produce Audit File.

Text Prompt File Input UI

The text prompt file input user interface will display the name and version of the program, explain the purpose of the program, and then ask for file input.

```
> Voting System - Version 1.0
> This system processes the winners of the election using either Open
Party Listing or Closed Party Listing Voting.
>
> Please enter the file name of the input file and press the [Enter]
key to submit:
>
```

Display Results UI

The display results user interface will display the winners of the election along with relevant statistics. This includes the winners, information about the election, the number of ballots cast, and stats for all candidates.

```
> Voting System - Version 1.0
> Election Results
>
> Election type: <OPL or CPL>
> Number of seats: <number>
> Ballots cast: <number>
>
> Winners: <List of winners>
>     Winner 1 <Party Affiliation>
>     Winner 2 <Party Affiliation>
>
> Candidate Statistics: <An alphabetized list of all candidates>
>     Candidate 1: <number> votes, <percent> of vote, <won/lost>
>     Candidate 2: <number> votes, <percent> of vote, <won/lost>
>     ...
```

Audit File UI

The audit file will be a CSV file that contains detailed information about the election results. This includes OPL/CPL, party number, ballot number, seat number, candidate name and affiliations (with candidates grouped by party), calculation of seat allocation, and seat winners and party affiliations. If it is OPL, the number of votes per candidate must also be displayed.

Voting System	Version 1.0	Detailed Election Results					
Election type	Number of seats	Ballots cast	Number of Parties				
<OPL or CPL>	<number>	<number>	<number>				
Parties	Candidates	Winner	If OPL, Votes				
<Party 1>	<Candidate 1>	<Y/N>	<nothing/number>				
	<Candidate 2>	<Y/N>	<nothing/number>				
<Party 2>	<Candidate 1>	<Y/N>	<nothing/number>				
	<Candidate 2>	<Y/N>	<nothing/number>				
	<Candidate 3>	<Y/N>	<nothing/number>				
<Party 3>	<Candidate 1>	<Y/N>	<nothing/number>				
Parties	Votes	First Allocation of Seats	Remaining Votes	Second Allocation of Seats	Final Seat Totals	% of Vote	% of Seats
<Party 1>	<number>	<number>	<number>	<number>	<number>	<number>	<number>
<Party 2>	<number>	<number>	<number>	<number>	<number>	<number>	<number>
<Party 3>	<number>	<number>	<number>	<number>	<number>	<number>	<number>

3.2 Hardware Interfaces

None

3.3 Software Interfaces

The input file received must be in a specific format depending on OPL/CPL. The input file is guaranteed to be correct. The input file will be in the same directory as the executable file. There is only one file per election.

For CPL

1st Line: CPL for closed party listing.
2nd Line: The number of seats.
3rd Line: The number of ballots.
4th Line: The number of parties.
Followed by the parties and their candidates in the ranked order of preference by the party.
Followed by all ballots.

```
CPL
3
9
6
Democratic, Joe, Sally, Ahmed
Republican, Allen, Nikki, Taihui
New Wave, Sarah
Reform, Xinyue, Nikita
Green, Bethany
Independent, Mike
1,,,,,
1,,,,,
,1,,,,
,,,,1,
,,,,,1
,,,1,,
,,,1,,
1,,,,,
,1,,,,
```

For OPL

1st Line: OPL for open party listing.
2nd Line: Number of seats.
3rd Line: Number of ballots.
4th Line: Number of candidates
Followed by the candidates and their party.
Followed by all ballots.

```
OPL
3
9
6
Democrat, Pike
Democrat, Lucy
Democrat, Beiye
Republican, Etta
Republican, Alawa
Independent1, Sasha
1,,,,,
1,,,,,
,1,,,,
,,,,1,
,,,,,1
,,,1,,
,,,1,,
1,,,,,
,1,,,,
```

3.4 Communications Interfaces

None

4. System Features

The use cases for the following features are in the Use Cases Document. A diagram detailing the actors involved in each use case can be found in Appendix B: UML Use Case Diagram.

4.1 Command Line File Input

4.1.1 Description and Priority

The command line file input is a method that the testers will use to input a file through the command line prompt. This feature is of medium priority because it allows testers to streamline input for testing purposes. This method of file input is not intended to be used by election officials and is therefore not the main file input, which is why it is of medium priority and not high priority.

4.1.2 Stimulus/Response Sequences

This system feature will trigger when the program is run with 1 command line argument. If there is more than one command line argument then the system will proceed with requirement Text Prompt File Input. If there is only 1 command line argument the system will extract the file name from the command line arguments. The system will then verify if the file exists, if it does not then the system will proceed with requirement Text Prompt File Input. If the file exists then the system will proceed with requirement Extract Info.

4.1.3 Functional Requirements

REQ-1.1: The system shall retrieve the file name through the command line.

REQ-1.2: The system shall start the Text Prompt File Input if the file name extracted from the command line does not exist.

REQ-1.3: The system shall start the Text Prompt File Input if there is more than one command line argument.

4.2 Text Prompt File Input

4.2.1 Description and Priority

The text prompt input is the main input method for the file name. This feature will primarily be used by election officials, who are the intended users of the program. This feature is a high-priority feature because it is the main method of file name input.

4.2.2 Stimulus/Response Sequences

The user will start the program and will be prompted to input a file name. The system will then extract the file name from the user input and verify that the file exists. If the file exists it will continue with the Extract Info use case. If the file does not exist, the system will prompt the user again. The user interface of the text prompt can be found in the User Interface Section: Text Prompt File Input UI, but the final form is TBD.

4.2.3 Functional Requirements

REQ-2.1: The system shall retrieve the file name through the text prompt.

REQ-2.2: The system shall re-prompt the user if the file does not exist in the system directory.

4.3 Extract Info

4.3.1 Description and Priority

Information fed into the voting algorithms must be read from a file. This will extract from the input file the voting type, as well as the number of ballots, parties, candidates, and seats. Also, information such as the names of parties and candidates as well as all of the ballots. This information is then stored for later use. This is a high priority because it is required to run the voting algorithm.

4.3.2 Stimulus/Response Sequences

There is no direct interaction from a user. The file name is attained from user input however, that is related to the requirements Command Line File Input and Text Prompt File Input, not this requirement. There is no direct output to the user, the information from this step is handled by the requirement CPL or OPL.

4.3.3 Functional Requirements

REQ-3.1: Information fed into the voting algorithms must be read from a file.

REQ-3.2: All information from the input file will be extracted and stored, including voting type, number of seats, ballots, parties, candidates, information about parties and candidates, and all of the ballots.

REQ-3.3: File IO errors that are encountered during this stage will be dealt with by closing the file, informing the user of the failure, and then restarting with use case Text Prompt File Input.

4.4 CPL

4.4.1 Description and Priority

The CPL algorithm is designed to handle elections that abide by the Closed Party Listing rules. Since this algorithm is essential to determine the winners, it necessitates a high priority.

4.4.2 Stimulus/Response Sequences

There are no direct user inputs for this action, the original user input was generated in the use cases Command Line File Input or Text Prompt File Input. There is direct no user output, the data from this phase is displayed in the use cases Produce Audit File and Display Results.

4.4.3 Functional Requirements

REQ-4.1: The CPL algorithm will be triggered if and only if the election is of type CPL, which is detailed in the first line of the input file.

REQ-4.2: The CPL algorithm must be followed correctly to determine the winners.

REQ-4.3: Any ties encountered during the algorithm must be solved with the Coin Flip requirement, and this must be fair.

4.5 OPL

4.5.1 Description and Priority

The OPL algorithm is designed to handle elections that abide by the Open Party Listing rules. Since this algorithm is essential to determine victorious candidates, it necessitates a high priority.

4.5.2 Stimulus/Response Sequences

There are no direct user inputs for this action, the original user input was generated in the use cases Command Line File Input or Text Prompt File Input. There is direct no user output, the data from this phase is displayed in the use cases Produce Audit File and Display Results.

4.5.3 Functional Requirements

REQ-5.1: The OPL algorithm will be triggered if and only if the election is of type OPL, which is detailed in the first line of the input file.

REQ-5.2: The OPL algorithm must be followed correctly to determine the winners.

REQ-5.3: Any ties encountered during the algorithm must be solved with the Coin Flip requirement, and this must be fair.

4.6 Produce Audit File

4.6.1 Description and Priority

This feature produces an audit file containing information on the election results including candidate names, party names, election type, seat winners along with their party affiliation, and the calculation for the largest remainder approach. This is a high-priority feature because this audit file is used to communicate the election results to the election officials. Moreover, this file is essential to establish a secure election procedure as it will be produced by the system directly without tempering.

4.6.2 Stimulus/Response Sequences

There are no user inputs for this action, the original user input was generated in the use cases Command Line File Input or Text Prompt File Input. This audit file will be created once the OPL and CPL algorithms finish tabulating ballots and have allocated all the seats to the respective election winners. The current format of the CSV file that is generated in this step can be found in the User Interfaces Section: Audit File UI, but the format is still a work in progress and will be refined in a later stage (TBD).

4.6.3 Functional Requirements

REQ-6.1: An audit file is generated and saved for viewing.

REQ-6.2: The audit file is assumed to be correct.

REQ-6.2: The audit file should outline which candidates/parties secured which seats.

REQ-6.3: The audit file should contain all election results information (See the Main Course UC_AUDIT for election results information).

4.7 Display Results

4.7.1 Description and Priority

After the results are calculated and the winners are determined, the results will be displayed on the screen in an easy-to-understand format. This must display the winners, information about the election, the number of ballots cast, and the statistics for all candidates. This is a high-priority requirement because it is the primary method of communicating the results of the election to the user.

4.7.2 Stimulus/Response Sequences

There are no user inputs for this action, the original user input was generated in the use cases Command Line File Input or Text Prompt File Input. There is a system response to the user that displays to the user the results of the election. An example UI is detailed in the User Interfaces section: Display Results UI, but it is not final and the exact format is TBD.

4.7.3 Functional Requirements

REQ-7.1: The winners that were calculated should be displayed on the screen.

REQ-7.2: The information about the election including the type of election and the number of seats should be displayed on the screen.

REQ-7.3: The number of ballots cast should be displayed on the screen.

REQ-7.4: The statistics for all candidates should be displayed on the screen. This includes those who did not win a seat. Specific statistics that must be included are the number of votes received and the percentage of votes received.

4.8 Coin Flip

4.8.1 Description and Priority

The coin flip feature is a step that occurs during the tabulation of ballots (OPL or CPL) to determine winners in the case that two or more candidates result in a tie. This feature is essential to the voting system and thus demands a high priority. See UC_COIN_FLIP in the Use Cases Document for further details.

4.8.2 Stimulus/Response Sequences

While OPL or CPL is run, the algorithms will attempt to allocate winners their seats. However, in this process, a tie could occur between any two or more candidates. If and only if this occurs, then a “coin flip” is enacted to handle the tiebreaker.

4.8.3 Functional Requirements

REQ-8.1: OPL or CPL must be actively running.

REQ-8.2: The algorithm should receive a signal that a tie has occurred and engage in a “coin flip”.

REQ-8.3: The results of the coin flip must be fair, and within a reasonable margin of error.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

The system must be able to run 100,000 ballots in under 4 minutes.

5.2 Safety Requirements

None

5.3 Security Requirements

The input file may not be modified in any way. When accessed, it must be read-only.

5.4 Software Quality Attributes

- Usability: There will only be one access point for all election types, that is, both CPL and OPL will be run using one program.
- Testability: The program will be easily testable using the command line.
- Reusability: The program will be able to be used at least once per election cycle.
- Correctness: The program will not provide invalid election results.

5.5 Business Rules

None

6. Other Requirements

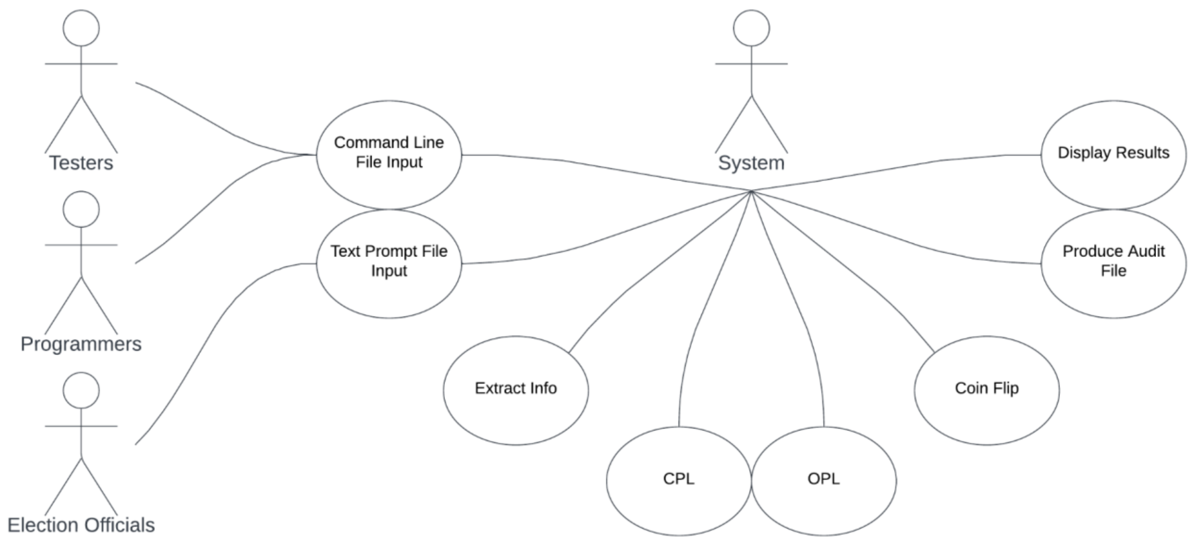
None

Appendix A: Glossary

- OPL (Open Party Listing): A voting method in which voters are permitted to vote only for an entire party, with candidates being chosen in a predetermined sequence to secure seats.
- CPL (Closed Party Listing): A voting method in which voters are permitted to vote for both a candidate and their affiliated party. The candidate selected, in the event of a party winning a seat, is the one who received the highest number of votes within their respective party.
- LRASA (Largest Remainder Approach to Seat Allocation): The approach that algorithms CPL and OPL use to determine the number of seats allocated to each specific party. See the Main Course of use case UC_CPL or UC_OPL in the Use Cases Document for a description of the LRASA.

Appendix B: Analysis Models

UML Use Case Diagram



Appendix C: To Be Determined List

- For the use case UC_COIN_FLIP, there is some more elaboration that is necessary to distinguish the two input cases from UC_CPL, however as there are minor differences in how UC_COIN_FLIP would be used (choosing which candidate gets a seat vs choosing which candidate goes first in the CPL order), to be elaborated in a later phase.
- The wording of our text user interface in the use case UC_INPUT_TEXT is not set and will be workshopped in a later phase.
- Similarly, the wording of our output user interface in the use case UC_DISPLAY is not set and will be workshopped in a later phase.
- Similarly, the wording of our audit file in the use case UC_AUDIT is not set and will be workshopped in a later phase. Also, ties are not expressed in the audit file right now, so that will be added along with the improved audit file format.
- The exact wordings of the various error messages that will be displayed during operation have not been set and will be created in a later phase.