Software Requirements Specification

for

OPL/CPL Voting System

Version 1.0 approved

Prepared by Connell Hagen (hage0686), Grant Oie (oiexx032),

Khalid Qasim(qasim009), and Michael Mulhall (mulha022)

CSCI 5801 Sec. 001

2/11/2024

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Group 13	2/7/2024	Initial draft	1.0
Group 13	2/11/2024	Final drafting and review	1.0

1. Introduction

1.1 Purpose

1.1.1 The purpose of this program is to calculate and verify the results of an open party list election and/ or a closed party list election. This is release number 1.0. The program will be able to be used, with different access levels as stated in the nonfunctional requirements, by testers, and election officials.. The program will take in a file with votes for various candidates, verify which kind of election is being run, execute the open party list or closed party list algorithm, allocate seats for candidates and parties, and return a results screen and audit file for the election. This is all contained within one directory, and the scope of this SRS is the entire program.

1.2 Intended Audience and Reading Suggestions

1.2.1 **Developers** - 1.3 gives an overview of the scope of the software and its purpose. All sections from section 2 through the end of the SRS contain helpful information for the outline of the requirements of the software, and its purpose.
1.2.2 **Project Managers** - 3.2 and 3.3 describes the environmental constraints that project workers must be able to effectively develop for. For other required information important for effective manager-worker interactions, look to the section of 1.2 for the type of worker being interacted with.
1.2.3 **Marketing Staff** - Sections 1.3 and 2.1 give a high-level overview of the full product, Section 2.2 and 4 list functions of the product that potential users will be interested in.
1.2.4 **Users** - 2.6 gives information about the availability of user documentation. Section 4 gives the available functions of the program, and the main courses to use each function.
1.2.5 **Testers** - 2.2 gives the functionality that must be tested thoroughly. 2.3.2 gives instructions for how to run tests on the system, along with the constraints from 2.4, 2.5 and 2.7. Use cases from section 4, along with sections 5.1, 5.2, and 5.3 give more specific details on what must be tested.
1.2.6 **Documentation Writers** - 2.6 gives information on the user documentation. 2.1 and 2.2 give a high-level overview of system interactions that must be documented, with section 4 giving more descriptive details pertaining to each function of the system.

1.3 Product Scope

This project entails the development of a voting system designed to implement and manage two distinct types of voting processes: open party list voting and closed party list voting. The system aims to offer a reliable, fair, and efficient mechanism for handling electoral data and determining election winners. The primary objectives include the accurate tabulation of votes, adherence to the principles of proportional representation, and ensuring a transparent and fair election outcome. Our vision is to create a user-friendly, secure, and versatile platform that supports various electoral frameworks and can be adapted to support different future electoral needs.

1.4 References

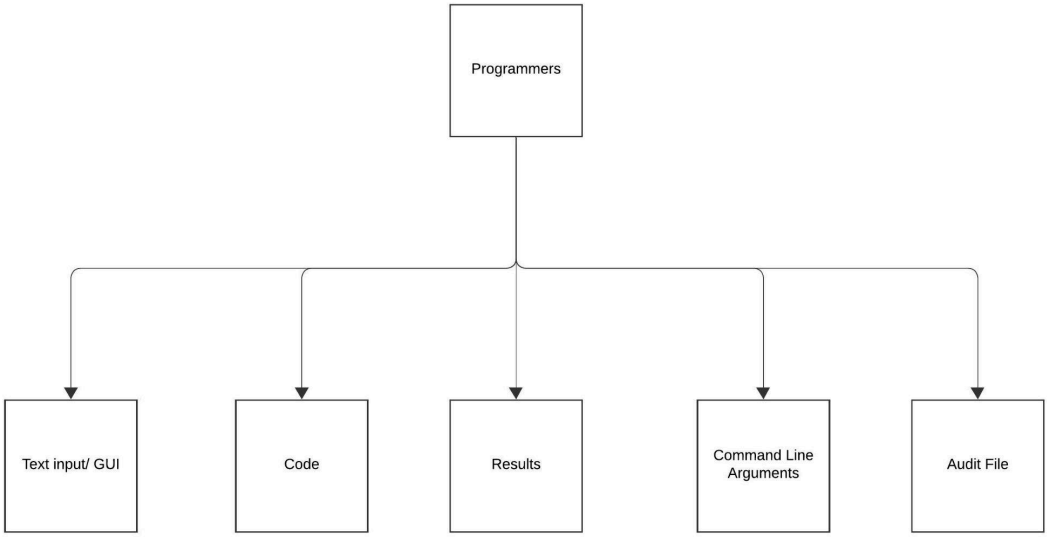
[1] CSCI 5801, Project 1 – Waterfall Methodology, Software Requirements Specification (SRS) Document for Voting System, Assignment Instructions

2. Overall Description

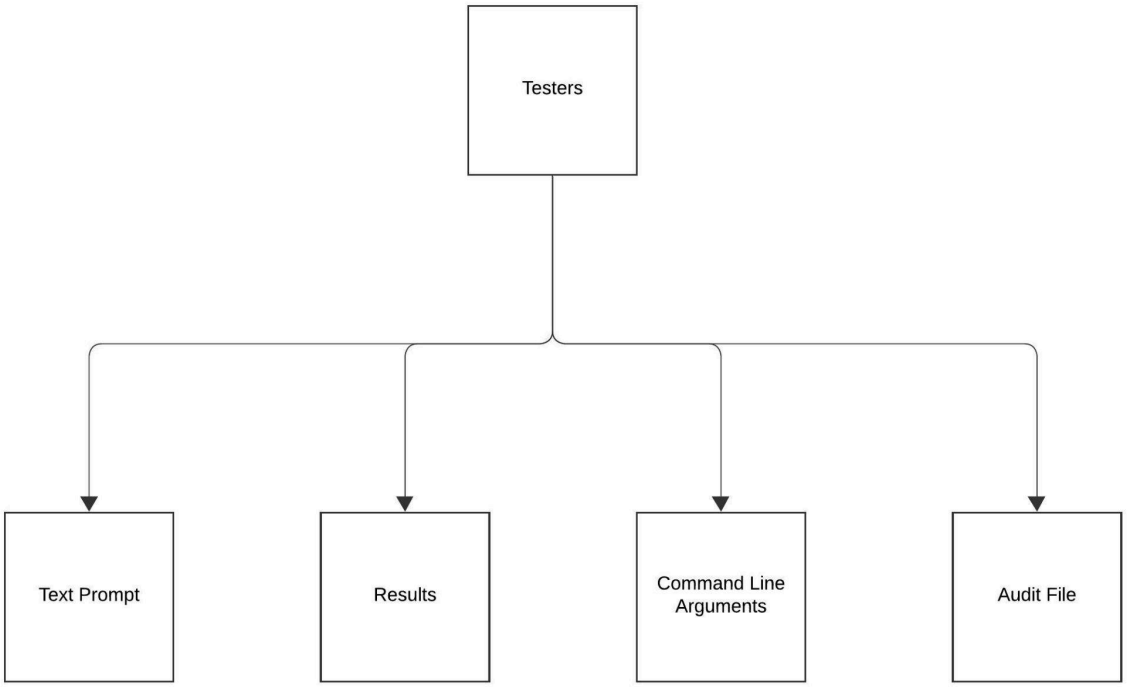
2.1 Product Perspective

2.1.1 This program is not a follow-on member of a product family. It is completely new and original. It is self-contained in one directory and has different access levels for different actors. The diagrams below show the relationships each actor has with different aspects of the program.

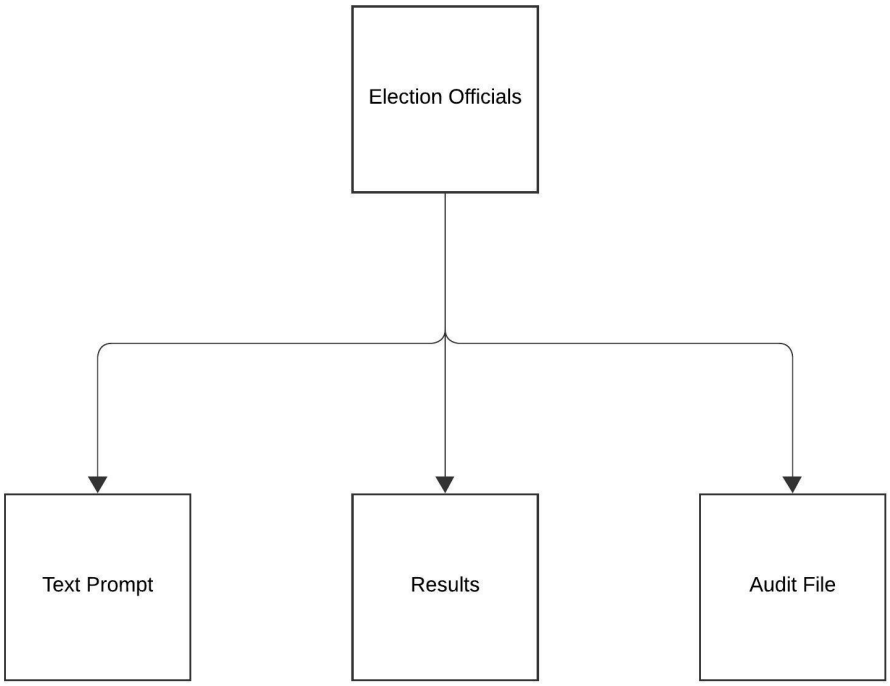
2.1.1.1 Programmer's perspective. Programmers will have access to everything in order to build the program, test it, and fix code if the Testers find issues.



2.1.1.2 Tester's perspective. The Tester will be able to use command line arguments, text prompt, audit file, and the results.



2.1.1.3 Election Official's perspective. Election officials will use a text prompt to input the ballot file and will receive the results and see the audit file



2.2 Product Functions

User-Interface

- Entering the filename as a command-line argument - The user is able to input the filename of the election data as a command-line argument
- Entering the filename to a terminal prompt - The user is able to initiate the program from the command-line with no arguments, then enter the filename into a terminal prompt from the program

Backend algorithms

- Data Parsing - The system converts election data within a file to data stored in a data structure that program algorithms can use.
- Largest Remainder Approach Seat Allocation - The system performs a largest remainder seat allocation algorithm for OPL and CPL ballot data.
- OPL - The system uses the results of the Largest Remained Approach Seat Allocation (see above), and assigns the won seats to individual candidates within the winning party who had the greatest number of votes.
- CPL - The system uses the results of the Largest Remained Approach Seat Allocation (see above), and assigns the won seats to individual candidates within the winning party who are first in the party's candidate listing.
- Tie Resolution - The system handles multi-party and multi-seat ties in a equitable and efficient manner, simulating a fair coin toss

Outputs

- Audit File Generation - The system generates an audit report file, detailing the type of election, number of parties, number of ballots, number of seats, the calculations for seat allocations, and a list of seat winners with their respective party affiliation.
- Displaying Election Results - The system displays the final election results, presenting winner(s) and information regarding the election

2.3 User Classes and Characteristics

2.3.1 Programmers

2.3.1.1 Programmers will have no security restrictions and will be able to access all of the code in order to build it and test it prior to handing it over to the Testers. The programmers responsibility is to build a working program for calculating the seat allocation, or results, for an OPL or CPL election.

2.3.2 Testers

2.3.2.1 Testers will be able to input command line arguments when testing the program. This will allow them to write scripts to test the code thousands of times over and guarantee its integrity before it is put into action in real elections.

2.3.3 Election Officials

2.3.3.1 Election officials will be able to start the program with a terminal command and then use the terminal text prompt for filename input. They will have no access to the code to prevent manipulation that could degrade the effectiveness of the product and produce an incorrect result. They will also have access to

the final results output and audit file to ensure the program ran correctly before passing the results on as official.

2.4 Operating Environment

The voting system is designed to operate in an environment compatible with CSE Lab Machines, which are equipped with the Ubuntu distribution of the Linux operating system. The voting system is intended to run on a standard desktop available in CSE Labs.

2.5 Design and Implementation Constraints

2.5.1 Policy Constraints: There are no policy constraints for this system.

2.5.2 Hardware Constraints: This software must be able to run on up-to-date CSE Labs Machines and must be able to process election results on 100,000 ballots in 4 minutes or less.

2.5.3 Technology Constraints: This system must be written in C++ or Java, and this system must be able to run on an x86_64 CPU architecture.

2.5.4 Language Constraints: This software must use English for any output that an actor may read.

2.5.5 Security Protocols: There are no required security constraints on the system.

2.5.6 Design Conventions: The system must be programmed in an object-oriented paradigm.

2.6 User Documentation

As per the requested features, there will be no documentation provided to the end users outside of standard course code commenting that will occur during the development process.

2.7 Assumptions and Dependencies

2.7.1 The format of the input file is specified as in [1].

2.7.2 There are no errors in the ballots. Each ballot will have one vote cast[1].

2.7.3 Users will use the most up to date CSE lab machines[1].

2.7.4 There are no numbering mistakes in the CSV file containing the ballots[1].

3. External Interface Requirements

3.1 User Interfaces

The voting system will feature a single user interface, the command-line interface, although there will be two different mechanism in which users interact with the program, depending on which type they are, Programmers and Testers, or Election Officials:

3.1.1 Programmer and Tester Interface

The command line interface allows for direct passing of election file names into the program as command-line arguments. The Command-Line Interface will offer programmers and testers the command-line navigation interface for running the voting system.

3.1.2 Election Official Interface

Upon starting the program the user (election officials) will be able to initiate the program from the command-line with no arguments, then enter the filename into a terminal prompt from the program

3.2 Hardware Interfaces

3.2.1 Supported Device Types

3.2.1.1 CSE Labs Machines, x86_64 CPU Architecture

3.2.2 Communication Protocols

3.2.2.1 All data transfer will be done locally; no protocols are necessary.

3.3 Software Interfaces

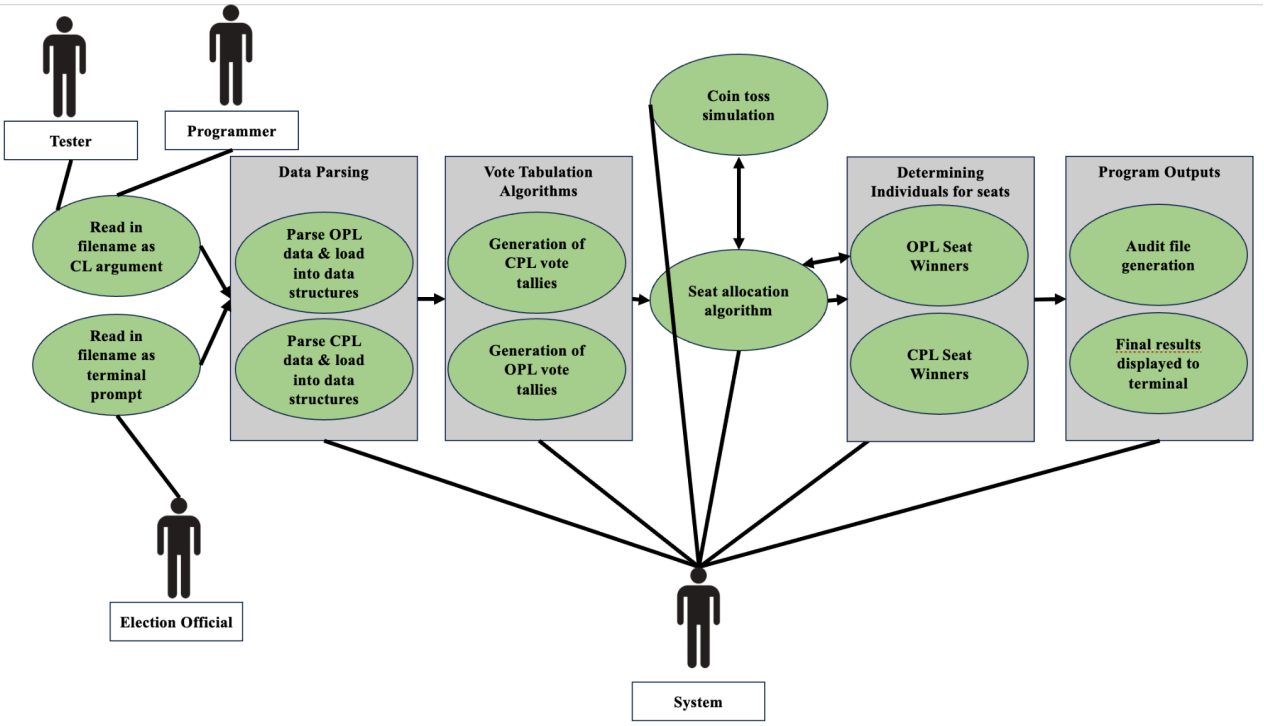
This program will only be verified for compatibility and usage on a University of Minnesota CSE Lab machine. The input interfaces include the terminal which will read in command-line execution of the program as well as the optional filename arguments. The program will also read in CSV files from reachable

directory pathways from the program location. The output interfaces will be in the terminal for printing election results, as well as a .txt file saved in the program's local directory for audit files.

3.4 Communications Interfaces

3.4.1 This program has no communications interfaces. It is entirely self-contained within its directory and can only create files and interact with files that are located in the directory.

4. System Features



The diagram illustrates the behaviors of the actors and the system. The diagram demonstrates inputs by both technical (programmers and testers) and non-technical (election officials) users, the processing of the system (data parsing, vote tabulation algorithms, seat allocation algorithm, and determining seat winners), and the program outputs of election results and an audit file. The actors for the voting system are defined below:

The System: This refers to the voting system being developed. The system processes election results from an input file containing election data. The system is responsible for parsing data into relevant data structures, vote tabulation algorithms for each type of election, the seat allocation algorithm, displaying election results, and generating an audit file. The system processes inputs from both technical users (programmers and testers) and non-technical users (election officials).

Programmers: The programmers are responsible for writing the program as specified by the functional requirements and the nonfunctional requirements. Programmers have access to all the code. These technical users interact with the voting system through command-line arguments to pass file names into the program.

Testers: The testers are responsible for ensuring the program will work every time. They are able to use the command line for arguments in order to use scripts to test more efficiently. These technical users interact with the voting system through command-line arguments to test the system's functionality handling various conditions.

Election Officials: These users interact with the voting system through a text prompt interface to enter the name of an input file for election file processing. Election officials will be able to input a file into the directory with the code and run the program through the instructions output to the terminal. They will then be able to see the results and confirm it with the audit file. Election officials will have no access to the code to prevent anyone from violating the integrity of the election.

4.1 Use Case 1 | Read in Filename as Command Line Argument

4.1.1 Use Case Analysis

Name	Read in filename as command line argument
ID	UC_001
Description	The user is able to input the filename of the election data as a command-line argument when running the program from terminal
Actors	Programmers, Testers
Organizational Benefits	Allows users to efficiently run the program on files from the command line. This enables users to set up automated testing processes.
Frequency of Use	Everytime the program is ran with a command-line argument, which can vary based on the requirements of testers and programmers
Triggers	The user wants to initiate the program
Precondition	● The file must be in a reachable directory from the program ● The user possesses a valid read-permissioned CSV file containing election data ● The file adheres to the predefined formatting specifications required by the program
Postcondition	● The program successfully reads the input filename, locates the file, and loads the data into the appropriate program data structures for downstream use by the program ● If the file is not found, or the format is incorrect, the user is notified with an appropriate error message
Main Course	1. The user initiates the program. 2. The system checks if a filename was provided as a command-line argument.

	If a filename is provided as a command-line argument: 3.1. The system attempts to open the file. (EX1,EX2) 3.2. The system imports the data into appropriate program data structures and proceeds to the next step of the process. (EX3) If no filename is provided as a command-line argument: 4.1 Proceed to use case 2 (ID:UC_002) The use case ends when the election data is successfully imported into the appropriate program data structures or the user decides to exit the program with an interrupt command. (A1)
Alternate Course	A1 ● If the user wishes to cancel the operation, they can issue an interrupt signal or close the terminal window.
Exceptions	EX1 ● Read Permission Denied: If the system lacks the necessary permissions to read the file, it alerts the user and requests the necessary access rights. EX2 ● File Not Found: If the system cannot locate the file, it informs the user and requests that the user tries again EX3 Incorrect File Format: If the file format is not a CSV, the system notifies the user and provides the format guidelines.

4.2 Use Case 2 | Read in Filename from a Terminal Prompt

4.2.1 Use Case Analysis

Name	Read in filename from a terminal prompt
ID	UC_002
Description	The user is able to initiate the program from the command-line with no arguments, the program then prompts the user to enter the filename of the election data into the terminal prompt
Actors	Election Official

Organizational Benefits	Provides users with the ability to initiate the program and then locate and enter the filename later on. This provides a more user-friendly approach than entering the filename via command-line argument
Frequency of Use	Everytime the program is ran without a command-line argument, which is a couple times per year outside of testing and development phases
Triggers	The user wants to initiate the program
Precondition	● The file must be in a reachable directory from the program ● The user possesses a valid read-permissioned CSV file containing election data ● The file adheres to the predefined formatting specifications required by the program
Postcondition	● The program successfully prompts for a filename, reads the input filename, locates the file, and loads the data into the appropriate program data structures for downstream use by the program ● If the file is not found, or the format is incorrect, the user is notified with an appropriate error message
Main Course	1. The user initiates the program. 2. The system checks if a filename was provided as a command-line argument. If a filename is provided as a command-line argument: 3.1 Proceed to use case 1 (ID:UC_001) If no filename is provided as a command-line argument: 4.1. The system prompts the user to enter the filename in the terminal. 4.2. The user inputs the filename.(A1) 4.3. The system attempts to open the file. (EX1,EX2) 4.4. The system imports the data into appropriate program data structures and proceeds to the next step of the process. (EX3) The use case ends when the election data is successfully imported into the appropriate program data structures or the user decides to exit the program with an interrupt command. (A2)
Alternate Course	A1 ● If the user provides an invalid filename or file path, the system prompts the user to re-enter the filename/filepath. A2 ● If the user wishes to cancel the program, they can issue a cancel command or close the terminal window.

Exceptions	EX1 ● Read Permission Denied: If the system lacks the necessary permissions to read the file, it alerts the user and requests the necessary access rights. EX2 ● File Not Found: If the system cannot locate the file, it informs the user and requests the correct file name in another terminal prompt. EX3 Incorrect File Format: If the file format is not a CSV, the system notifies the user and provides the format guidelines.

4.3 Use Case 3 | Resolving a Tie in the Election with a Fair Coin Flip

4.3.1 Use Case Analysis

Name	Resolving a tie in the election with a fair coin flip
ID	UC_003
Description	If there is any tie in the election results, the program will perform a simulated coin flip to determine the winner. The occurrence of a tie and the resulting winner will be detailed in the audit file as included in the final terminal output
Actors	System
Organizational Benefits	Ensures a transparent, unbiased, and swift resolution to tied election results, maintaining the integrity and efficiency of the process.
Frequency of Use	As needed, depending on the occurrence of a tie in the election results, which is generally rare
Triggers	A tie is detected during the seat allocation algorithm
Precondition	● The election data has been read into the system ● The CPL or OPL algorithms have been performed ● A tie is checked for and detected during the seat allocation algorithm
Postcondition	The tie is resolved using a fair and random method, a simulated coin-flip, and the winner is recorded. The details of the tie resolution process are logged in the audit file and presented in the terminal output for transparency and record-keeping.
Main Course	1. The system identifies a tie between two or more candidates during the seat allocation algorithm 2. The number of parties involved in the tie is determined

	a. If the number of parties is greater than two, go to A1, then return to step 4 of the Main Course b. If the number of parties is equal to 2, continue with step 3 of Main Course 3. The system initiates the tie-resolving process by simulating a fair coin flip a. If the parties receive a 1 during a single simulation, this is considered a 'win' b. The coin flip simulation will be ran 1001 times, and the majority winner of that set of simulations will be the ultimate winner 4. The winner of tie process is allocated a seat and if there were multiple seats in contention during the tie, the winner is removed from the tied party set, and the remaining tied parties, along with the decremented number of seats return to step 1 of Main Course 5. The details of the tie, the coin flip process, and the chosen winner are recorded in the audit file 6. The system updates the election results to reflect the winner of the tie 7. The final results, including the resolution of the tie, are detailed in the audit file and displayed in the final terminal output. 8. The use case ends when the tie is broken, seats are allocated, and all necessary records are updated and saved
Alternate Course	A1 ● If the tie involves more than two parties (parties = n), there is an n-way coin flip simulation where those members who get a 1 in each roll receives a point. If multiple parties receive a 1, the process is paused and returns to the beginning of A1 until a winner of that sub-tie is determined. This simulation is ran 1001 times, and the party with the most wins is the ultimate winner of the tie.
Exceptions	There are no exceptions for this use case.

4.4 Use Case 4 | Parsing of OPL Data and Loading into Data Structures

4.4.1 Use Case Analysis

Name	Parsing of OPL Data and Loading into Data Structures
ID	UC_004
Description	On a set of OPL ballot data, parse the data and store it in representative data

	structures for OPL ballot data.
Actors	System
Organizational Benefits	Encapsulates the conversion of raw data into data structures that algorithms can be run on.
Frequency of Use	Once per OPL election for which the system is used to tabulate.
Triggers	An input file has been received, and it is determined to be for an OPL election.
Precondition	● Either UC_001 or UC_002 has been successfully completed, and all of its respective preconditions remain true. ● The input file's header denotes that it is an OPL election.
Postcondition	● The data will be accurately parsed, and the data will be stored in 2 data structures ● 1 data structure will hold data of the party names and total votes for each party ● 1 data structure will hold data of the candidate names, and their respective party and number of votes
Main Course	1. The system determines that an input file of election results represents an OPL election 2. The system reads through the input file once, starting after the header 3. The system records necessary data into an accessible data structure
Alternate Course	There are no alternate courses for this use case.
Exceptions	There are no exceptions for this use case.

4.5 Use Case 5 | Parsing of CPL Data and Loading into Data Structures

4.5.1 Use Case Analysis

Name	Parsing of CPL Data and Loading into Data Structures
ID	UC_005
Description	On a set of CPL ballot data, parse the data and store it in a representative data structure for CPL ballot data.
Actors	System
Organizational Benefits	Encapsulates the conversion of raw data into a data structure that an algorithm can be run on.
Frequency of Use	Once per CPL election for which the system is used to tabulate.

Triggers	An input file has been received, and it is determined to be for a CPL election.
Precondition	● Either UC_001 or UC_002 has been successfully completed, and all of its respective preconditions remain true. ● The input file's header denotes that it is a CPL election.
Postcondition	● The data will be accurately parsed, and the data will be stored in 2 data structures ● 1 data structure will hold data of the party names and total votes for each party ● 1 data structure will hold data the candidate names in their proper order
Main Course	1. The system determines that an input file of election results represents an CPL election 2. The system reads through the input file once, starting after the header 3. The system records necessary data into an accessible data structure
Alternate Course	There are no alternate courses for this use case.
Exceptions	There are no exceptions for this use case.

4.6 Use Case 6 | Performing Seat Allocation for OPL and CPL Elections

4.6.1 Use case Analysis

Name	Perform largest remainder approach seat allocation algorithm for OPL and CPL data
ID	UC_006
Description	This use case describes the process of allocating seats based on the largest remainder approach using the ballot data loaded from the Open Party List and Closed Party List elections. This process involves calculating quotas, initially allocating seats to parties by share of votes, and then allocating the remaining seats according to the largest remainder.
Actors	System
Organizational Benefits	Enables fair and efficient seat allocation for elections, ensuring accuracy for voters.
Frequency of Use	This process is executed after ballot data for an OPL or CPL election has been parsed and stored in representative data structures, when determining seat allocation would be necessary.
Triggers	Parsing data and loading into data structures is complete (see UC_004 & UC_005)

Precondition	Data parsing complete and data stored in representative data structures
Postcondition	All seats are allocated according to the largest remainder approach and seats have accurately been assigned to parties
Main Course	1. The system establishes a quota based on the total number of seats available. A quota is calculated by dividing total votes casted by total seats available. 2. The system computes initial seat distributions to each party. The quota is divided by the votes each party receives, and parties receive a seat for each whole number produced. 3. After the first allocation, the system computes remaining votes for each party by performing remainder division between their total vote count and the quota 4. The system allocates the remaining seats by party with the most remaining votes. Remaining votes are compared for each party, and the parties with the largest remainders are allocated the remaining seats (for ties, see A1; for parties who will be assigned more seats than candidates, see A2) 5. The system finalizes seat allocation for parties
Alternate Course	A1: The election must assign less seats than there are tied parties ● The tie is resolved using a fair coin flip (see UC_003) A2: A party has already been assigned as many seats as they have candidates ● The party is eliminated for consideration of more seats in the future, and all candidates they are running will get a seat.
Exceptions	There are no exceptions for this use case.

4.7 Use Case 7 | Choosing of OPL Seat Winners

4.7.1 Use Case Analysis

Name	Choosing of OPL Seat Winners
ID	UC_007
Description	Using the number of determined seats by party, assign the seats to specific candidates.
Actors	System
Organizational Benefits	Fairly determines the specific winners of seats within each party.
Frequency of Use	Once per OPL election for which the system is used to tabulate.

Triggers	Allocation of seats (see UC_006) has completed
Precondition	● UC_006 has been successfully completed. ● No party has been assigned more seats than they have candidates.
Postcondition	● The results will determine the correct winning candidates in every deterministic scenario. ● Winners will be fairly picked in the event of a tie.
Main Course	1. The system receives the number of allocated seats per party 2. For each party assigned X seats, the system chooses the top X candidates with the most votes within the party (for ties, see A1).
Alternate Course	A1: The election must pick less seat winners than there are tied candidates ● The tie is resolved using a fair coin flip (see UC_003)
Exceptions	There are no exceptions for this use case.

4.8 Use Case 8 | Choosing of CPL Seat Winners

4.8.1 Use Case Analysis

Name	Choosing of CPL Seat Winners
ID	UC_008
Description	Using the number of determined seats by party, assign the seats to specific candidates.
Actors	System
Organizational Benefits	Fairly determines the specific winners of seats within each party.
Frequency of Use	Once per CPL election for which the system is used to tabulate.
Triggers	Allocation of seats (see UC_006) has completed
Precondition	● UC_006 has been successfully completed. ● No party has been assigned more seats than they have candidates.
Postcondition	● The results will determine the correct winning candidates in every deterministic scenario.
Main Course	1. The system receives the number of allocated seats per party 2. For each party assigned X seats, the system chooses the first X candidates within the party's candidate list.

Alternate Course	There are no alternate courses for this use case.
Exceptions	There are no exceptions for this use case.

4.9 Use Case 9 | Generation and Output of Audit Report

4.9.1 Use case analysis

Name	Generation and Output of Audit Report
ID	UC_009
Description	This use case describes the process for the automatic generation of audit reports after the completion of an election using the voting system. The audit report details information regarding each election including type of election (OPL or CPL), number of parties, number of ballots, number of seats, the calculation for the largest remainder approach to seat allocation, and a list of seat winners with their respective party affiliations.
Actors	System
Organizational Benefits	Ensures accountability in the election process by providing a detailed record of election results and calculations used.
Frequency of Use	Executed at the completion of each election cycle the voting system is used.
Triggers	Seat winners selected
Precondition	Seat winners selected and election results finalized
Postcondition	An audit report is generated and made available to election officials
Main Course	1. The system compiles election data, including election type, vote totals, candidate and party information, and seat allocation calculations. 2. The system generates an audit report with the compiled data 3. The system saves the audit report in a .txt file in local directory, accessible to election officials
Alternate Course	There are no alternate courses for this use case.
Exceptions	There are no exceptions for this use case.

4.10 Use Case 10 | Display of Election Results

4.10.1 Use Case Analysis

Name	Display of Election Results
ID	UC_010
Description	This use case describes displaying the final election results, presenting the winner(s) and information regarding the election. This display will include the number of ballots cast, type of election, winners, and additional statistics relevant to the election process. Results are displayed in a user-friendly format accessible to election officials.
Actors	System
Organizational Benefits	Provides immediate access to comprehensive election results, ensuring accountability in the election process.
Frequency of Use	Executed at the completion of each election cycle the voting system is used.
Triggers	Seat winners selected
Precondition	Seat winners selected and election results finalized
Postcondition	Election results are successfully displayed
Main Course	1. The system calculates final election statistics 2. The system compiles and formats final election results and additional relevant statistics. 3. The system presents the formatted results on a screen accessible to election officials
Alternate Course	There are no alternate courses for this use case.
Exceptions	There are no exceptions for this use case.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

5.1.1 The results for 100,000 ballots will be output in 4 minutes or less.

5.1.1.1 Rationale: The election officials need to be able to receive the results of an election in a timely manner. Elections deal with large populations, so the program needs to be able to process a large amount of information quickly.

5.1.2 Written in C++ or Java.

5.1.2.1 Rationale: This program cannot be written in something other than an object oriented programming language in order for it to function properly. Writing the program in C++ gives the programmers flexibility in being able to efficiently read/ write to other files, take in information from the user, like the ballot file, and the program can inherit abilities from other programs if necessary.

5.1.3 Runs on CSE lab machines.

5.1.3.1 Rationale: In order to get a consistent runtime and execution of the code the program must be run on CSE lab machines. These machines all have the same specifications and hardware, where running this program on other machines will not guarantee timely results and the code working efficiently.

5.1.4 1 program for both election types

5.1.4.1 Rationale: 1 program for both OPL and CPL elections gives the program the ability to step by step figure out the type of election, which will be stated in the input file anyway, and then choose which algorithm to run before allocating seats. If the program was split into two different files, 1 for each election type, a level of complexity or redundancy would be added when either choosing to run 1 file over the other or running both and ignoring the output of the file with the incorrect election type.

5.1.5 Ballot file in the same directory as the program

5.1.5.1 Rationale: The ballot file being in the same directory as the program simplifies the reading process of the program. This is when the program takes in the data from the ballot file and begins to process it. If the program had to look outside of the directory for this file it would make the reading process more difficult, and the process of getting the proper file path more difficult for the users.

5.1.6 File with votes is a CSV file

5.1.6.1 Rationale: CSV is a common file type that separates data into commas. This creates something similar to a table format. The program will be able to read the data in a CSV file more easily due to its organized structure. This eliminates the possibility of reading the votes incorrectly, if a properly formatted CSV file is read by the program.

5.1.7 Makefile to compile program

5.1.7.1 Rationale: Programs written in C++ require something called a Makefile to run more quickly. A Makefile compiles all of the code in one go, which eliminates the steps for compiling every file individually before running. This speeds up the process of executing the program.

5.1.8 Program uploaded to Github.

5.1.8.1 Rationale: Github provides something called version control. Version control is where different people are able to make changes at different times to different things to contribute to the finished program. This provides flexibility to the programmers where they will be able to code on different machines at different times.

5.2 Safety Requirements

5.2.1 Election officials will not be able to access the code.

5.2.1.1 This would be a breach of the integrity of the election results that are output. Election officials will only be able to run the code with the csv file that contains the ballots.

5.3 Security Requirements

5.3.1 There are no security requirements for actions taken by those who are not an actor or user.

5.4 Software Quality Attributes

5.4.1 Portability

5.4.1.1 The program will be able to be run on any up to date CSE lab machine. Election officials are able to use any of these machines for a correct, speedy calculation. Testers must also use CSE lab machines to test the code.

5.4.2 Availability

5.4.2.1 Any election official will be able to procure this program as long as they have access to the most up to date CSE lab machines. Various actors will also be able to access the program at different levels. Programmers will be able to manipulate the code and Testers will be able to test it using command line arguments..

5.4.3 Testability

5.4.3.1 Testers will be able to write scripts to test the code as many times as possible to ensure its validity.

5.4.4 Reliability

5.4.4.1 The program will be able to output a correct result every time. This is assuming the csv file is formatted correctly. In the case of a tie the program will be reliably random as stated in 1.4.4.

5.4.5 Reusability

5.4.5.1 This program will be able to be used multiple times per year for OPL and CPL elections of any size. Should a new type of election be added to the program in the future, having a system that uses one program for all types of elections allows for code reuse.

5.4.6 Flexibility

5.4.6.1 The program is able to produce a correct result for both OPL and CPL elections. By allowing for multiple types of elections to work with the same one program, the software is able to flexibly handle different use cases.

5.5 Business Rules

5.5.1 Personnel

5.5.1.1 Tester: The testers are responsible for ensuring the program will work every time. They are able to use the command line for arguments in order to use scripts to test more efficiently.

5.5.1.2 Programmer: The programmers are responsible for writing the program as specified by the functional requirements and the nonfunctional requirements. Programmers have access to all the code

5.5.1.3 Election Officials: Election officials will be able to add the file containing the ballots into the directory with the code and run the program through the instructions stated in the text prompt. They will then be able to see the results and confirm it with the audit file. Election officials will have no access to the code to prevent anyone from violating the integrity of the election.

5.5.2 Functions

5.5.2.1 OPL Algorithm: The OPL algorithm will only be run if the system verifies, through the header of the csv file input by the election official, that the election type is OPL.

5.5.2.2 CPL Algorithm: The CPL algorithm will only be run if the system verifies, through the header of the csv file input by the election official, that the election type is CPL.

6. Other Requirements

Appendix A: Glossary

Open Party Listing(OPL): A type of election where voters express interest in a party based on the candidate they vote for. Seats are allocated based on the party that is voted for the most and then the candidate of the party that received the most votes.

Closed Party Listing(CPL): A type of election where parties rank their preferred candidates and voters vote only for a party. Seats are allocated based on the percentage of votes each party receives and the ranking of candidates each party creates.

Actor: Anything that interacts with the program.

Functional Requirement: A function of the system defined by an input/output interaction.

Nonfunctional Requirement: A constraint on the system that has been specified by the customer.

Use case: A description of how actors will perform tasks within the system.

Algorithm: A process that takes an input and generates an output.