Team 8
Election System
Software Design Document

Name (s): Samuel Thorson, Austin Odusanya, Daniel Ginzburg, Isaac Wodele
Lab Section: 008
Workstation:

Date: (04/03/2025)

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

This software design document describes the architecture and system design of the Election System. It is intended to be viewed by software developers as a guide to the development process or testers that directly interact with the system.

1.2 Scope

The Election System is a standalone system meant to determine election results from a CSV file. By providing the file name, number of seats in the election, and the type of election, election officials can use the system to calculate the winners. The system supports two types of elections: single transferable vote (STV) and plurality. The benefit of this system is that it allows election officials to quickly and accurately determine election winners.

1.3 Overview

This document provides an overview of the design of the election system. Section 2 provides a more detailed overview of the system. Section 3 describes the system architecture design (3.1) as well as a decomposition of the classes that make up the system (3.2-3.3). Section 4 explains how data is transformed and stored within the system (4.1-4.2). Section 5 details the individual components that make up the system. Section 6 displays the design for the user interface. Section 7 is a requirement matrix that maps system components to the requirements of the system. The appendix gives additional information on the implementation of the system.

1.4 Definitions and Acronyms

GUI: Graphical user interface

CLI: Command line interface

STV: Single transferable vote

CSV: Comma-separated values

Droop Quota: is the minimum number of votes a party or candidate needs to receive in a district to guarantee they will win at least one seat

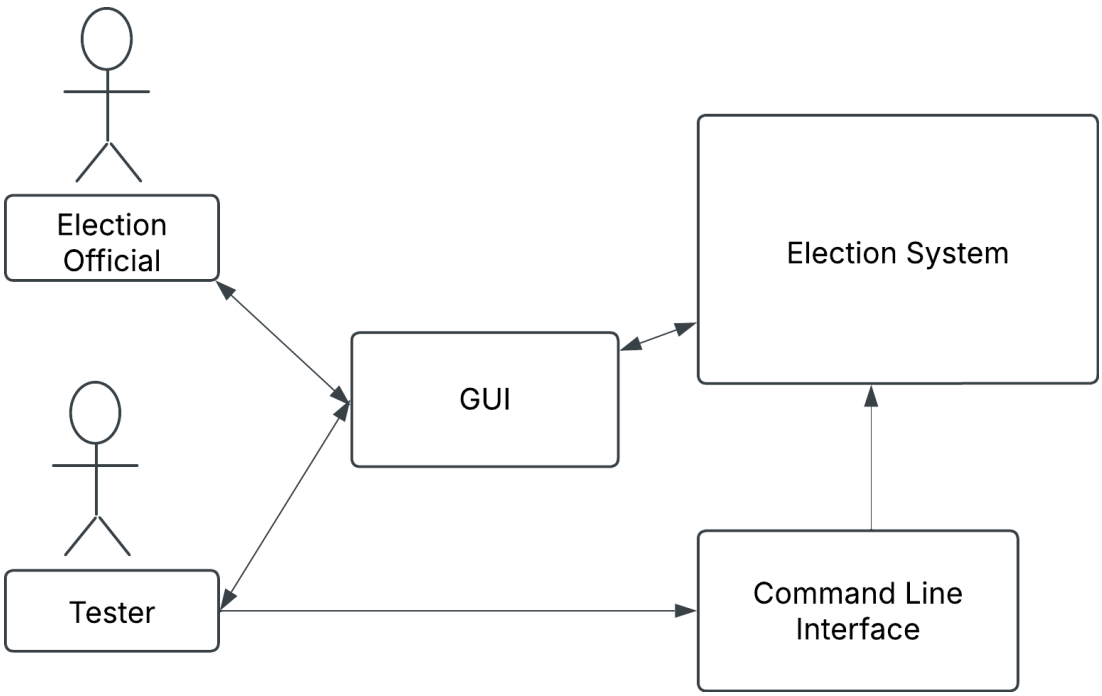
2. SYSTEM OVERVIEW

This product is a standalone system made to aid election officials in calculating election results. It

is used by both election officials and testers in a local environment. Both the election officials and testers access the program through the graphical user interface (GUI). The tester also accesses the program directly through the command line interface (CLI).

Ballots are passed into the system through a CSV file, where they will eventually be organized into ballot objects for the use of the system. Candidates will also be read from the file and brought into candidate objects so that they may be accessed later. Based on the voting algorithm chosen by the user, these ballot objects are read by the system in order to determine the winners for a given election.

The output of a given election is delivered through the GUI, where a user can request an audit file to print the results of the election.

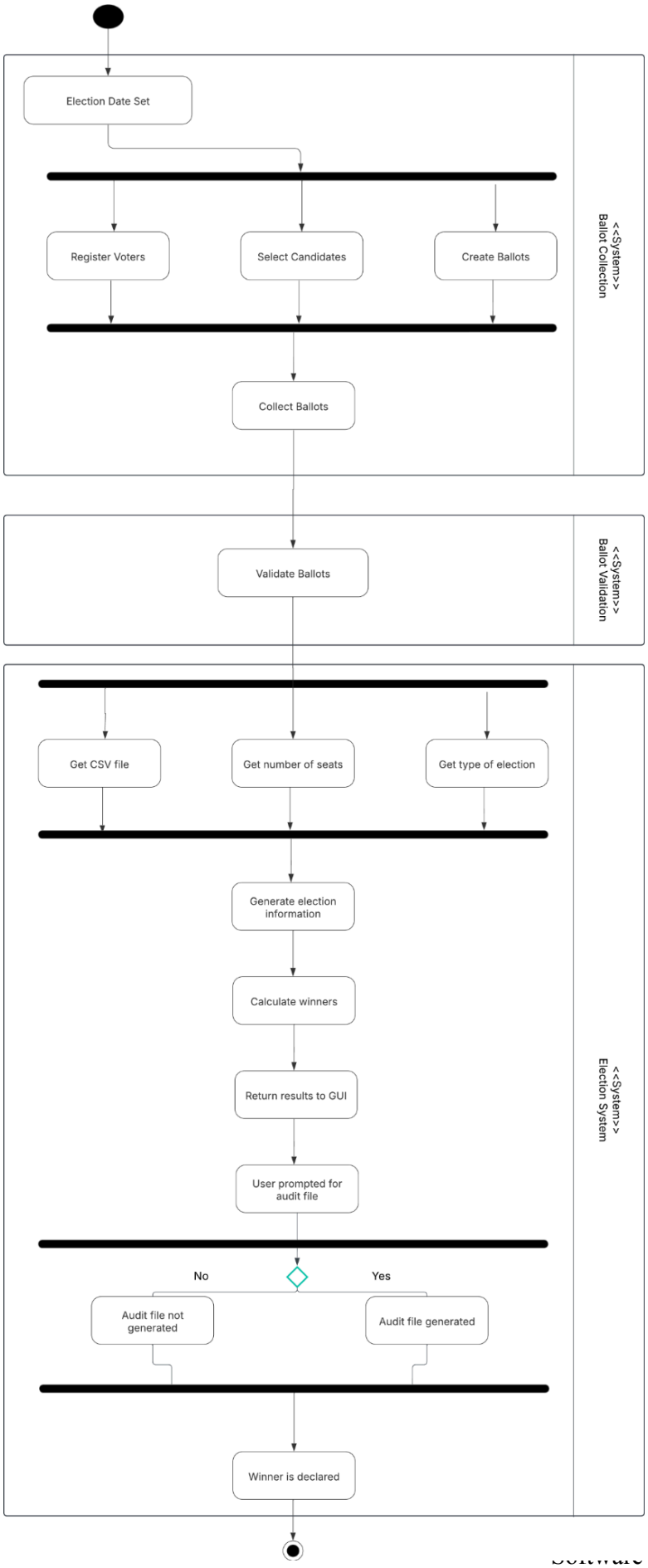


3. SYSTEM ARCHITECTURE

3.1 Architectural Design

The election system is split into three modules: ballot collection, ballot validation, and the election system component. The modules work together in this order to produce accurate and efficient results. The first module is ballot collection. Collection encompasses all of the activities that occur before the election such as setting the date and creating the ballots. The next module, ballot validation, makes sure that the submitted ballots have been filled out correctly and can be counted. After the ballots are collected and validated, the system can begin to total the election results. The system will be able to handle both single transferable voting (STV) and plurality elections, outputting the correct results.

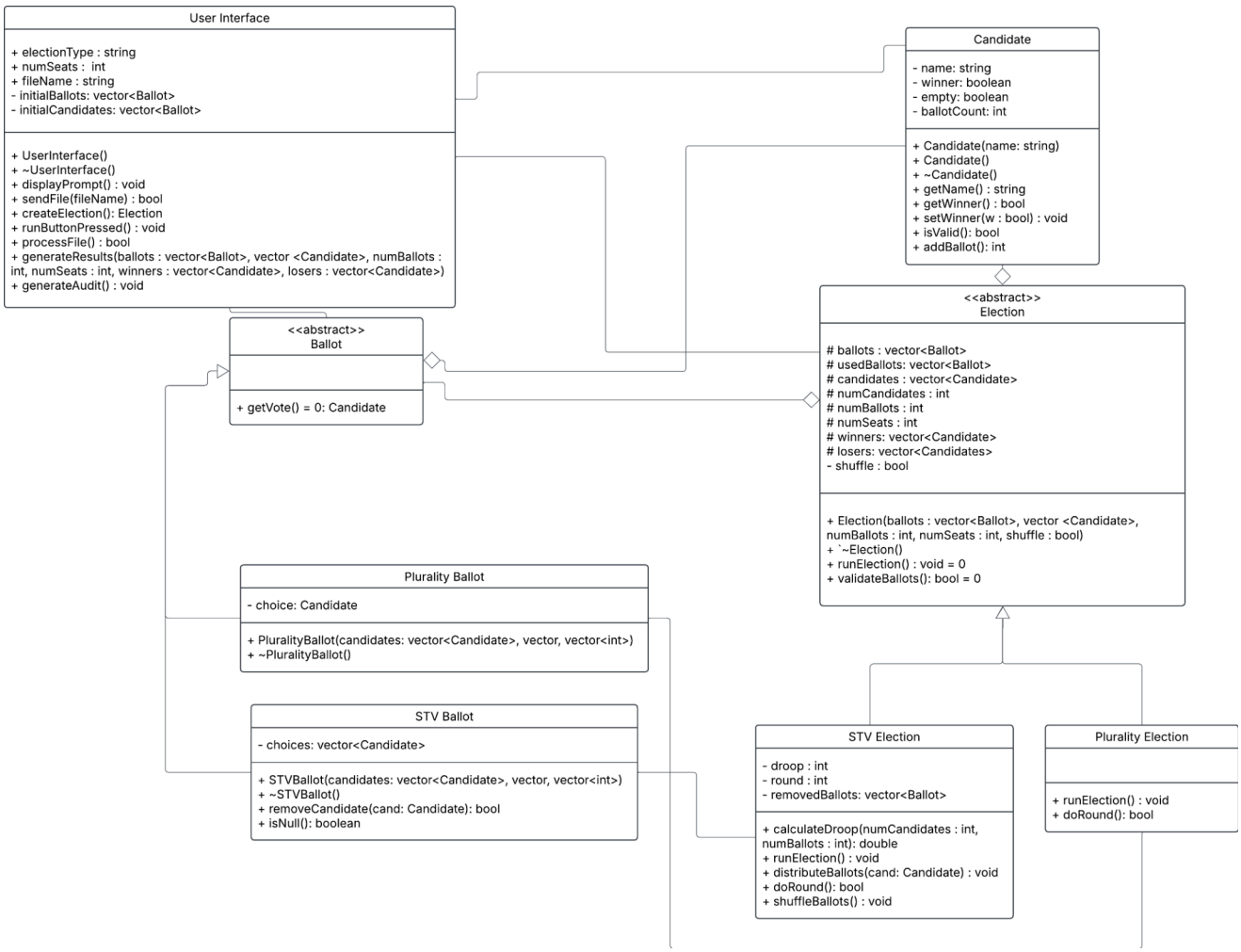
The diagram below represents the three modules interacting for a plurality election.


Election Date Set
Register Voters
Select Candidates
Create Ballots
Collect Ballots
<<System>>
Ballot Collection
Validate Ballots
<<System>>
Ballot Validation
Get CSV file
Get number of seats
Get type of election
Generate election information
Calculate winners
Return results to GUI
User prompted for audit file
No
Yes
Audit file not generated
Audit file generated
Winner is declared
<<System>>
Election System

3.2 Decomposition Description

The election and ballot classes were made abstract to be implemented for STV or plurality uses. The candidates will be created as individual objects and used in the election and ballot classes to organize data storage. The user interface will be how information is entered into the system and subsequently run the election.

This design allows the election classes to handle ballot validation and counting of votes. Collecting the ballots happens outside of the system.

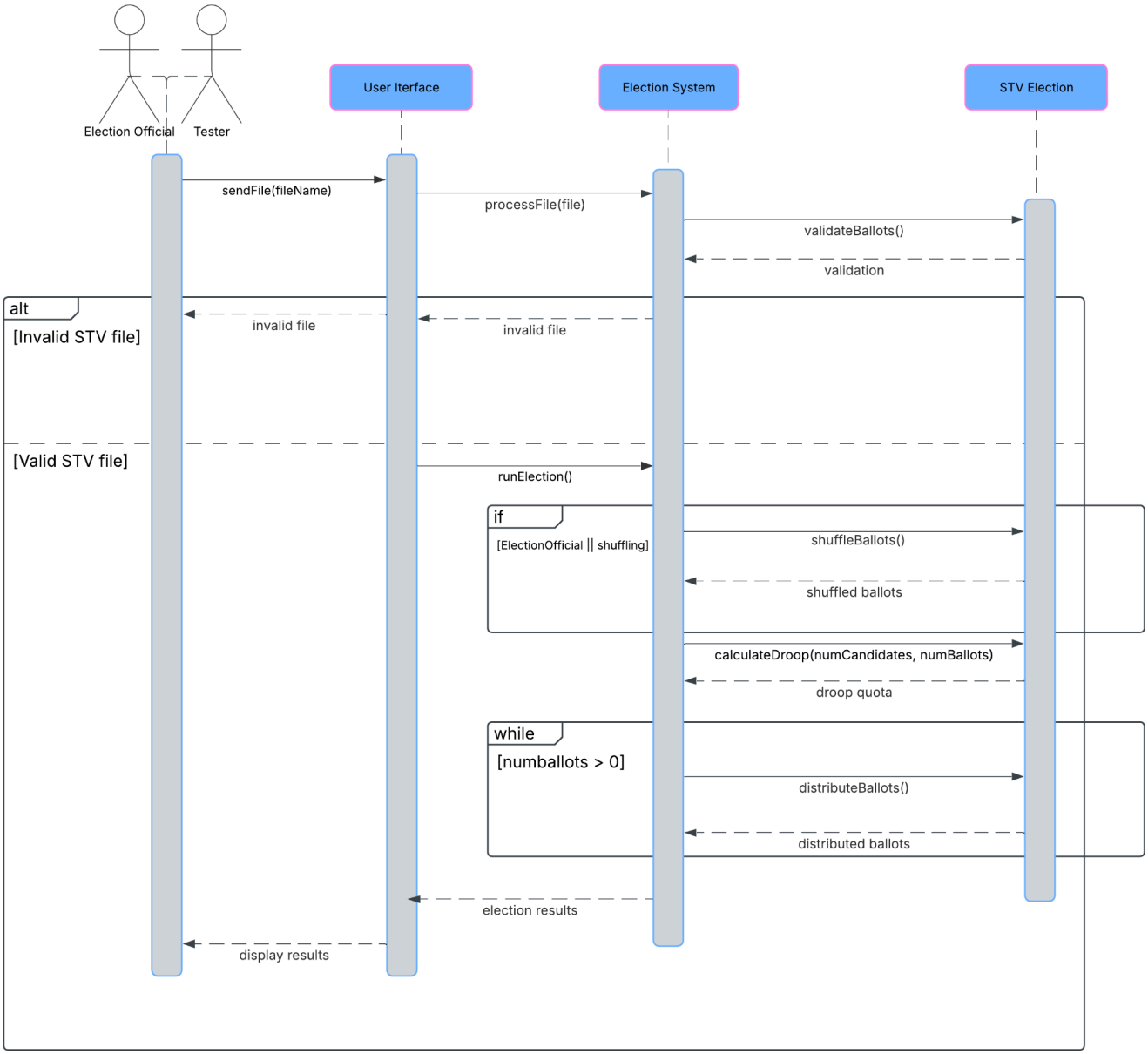


3.3 Design Rationale

The chosen architecture allows us to minimize complexity in the system while still maintaining a dependable process. Ballot collection and ballot validation occur before the Election System's process to ensure that any ballots coming into the system are processed efficiently. Validation also allows us to assume that there are no security concerns that the Election System must handle.

Ballots are processed into STV or Plurality ballot objects to allow the corresponding election class to count ballots efficiently. The User Interface class serves as the access point for users and election officials to use the system with ease. Candidate objects are created to keep track of winners throughout the process so they may be shown once the program has completed processing the election.

We considered including a ballot validator in the system to confirm that ballots could be processed by the system properly; however, we concluded that this part of the system was unnecessary due to the Ballot Validator system. Instead, the user interface class will show an error message if the CSV file is unable to be processed into ballot and candidate objects.



4. DATA DESIGN

4.1 Data Description

When an election official inputs a CSV file for processing, the ballots and candidates in the file are loaded into separate lists (vectors) that allow for iterative processing. Ballots can fall under two types, STV or Plurality. STV ballots consist of a vector of Candidates in the order of their vote while plurality ballots consist of just a single candidate indicating the single vote. Candidates are simply an object consisting of a string representing their name and a boolean indicating if they won a seat or not. There are no other external databases utilized in the election system.

4.2 Data Dictionary

Election:

Attribute/Method	Type	Description
ballots#	vector<Ballot>	stores all ballots
candidates#	vector<Candidates>	stores all candidates
fileName-	string	name of CSV file
generateAudit(filename : string)+	void	creates audit file
numCandidates#	int	number of candidates
numSeats#	int	number of seats
runElection()+	void	runs the election process
shuffle-	bool	indicates if ballots shuffled
validateBallots()	bool	validates if ballots match election information

Plurality Ballot:

Attribute/Method	Type	Description
choice-	Candidate	Candidate who gets ballot
getVote()+	Candidate	returns Candidate of a given ballot

STV Ballot:

Attribute/Method	Type	Description

choices-	vector<Candidate>	stores Candidates in ballot
getVote()+	Candidate	returns Candidate of a given ballot
isNull()+	bool	returns if ballot is empty
removeCandidates(cand: Candidate)	void	removes candidate from ballot

Candidate:

Attribute/Method	Type	Description
name-	string	name of Candidate
winner-	bool	candidate wins seat
getName()+	string	returns Candidate name
getWinner()+	bool	returns winner status
setWinner(w : bool)	void	sets winner status of candidate

5. COMPONENT DESIGN

5.1 User Interface

- displayPrompt(): void
    - Display the initial prompt to the screen that requests for the user to enter the election type, ballot list filename, and the number of seats.
- sendFile(fileName): bool
    - Determines whether the filename entered exists. Stores the filename if it does exist. Returns whether or not the filename exists.
- processFile(): bool
    - Opens saved file, attempts to read it as a csv, attempts to get and save initial Candidate vector and create an initial ballot vector. Returns whether or not the process was successful.
- createElection(): Election
    - Create a new election subclass object based on the election type, seat count, and the processed file. Returns the created election.
- generateResults(ballots : vector<Ballot>, vector <Candidate>, numBallots : int, numSeats : int, winners : vector<Candidate>, losers : vector<Candidate>): void

    - Display the results of the election to the GUI based off of the data after the election algorithm.
- generateAudit(filename): void
    - Called when ‘Generate Audit’ is clicked on results displayed on the GUI. Creates an audit file named filename that details the results of the election.
- runButtonPressed(): void
    - Called when ‘Run’ is clicked on the initial prompt on the GUI. Creates an election, runs the algorithm, then generates results based on election object data.

5.2 Ballot

- getVote() = 0: Candidate
    - Pure Virtual. Returns the Candidate that the ballot currently represents. Returns null Candidate if not representing anything.

5.2.1 STV Ballot

- removeCandidate(cand: Candidate): bool
    - If a candidate is included in the choices list, remove the candidate. No-ops otherwise. Returns true if the current vote of this ballot is the removed candidate, otherwise false.
- isNull(): boolean
    - Returns whether or not there are any candidates left represented in this ballot.

5.2.2 Plurality Ballot

No specific new methods.

5.3 Candidate

- Candidate()
    - Constructor representing a null Candidate.
- Candidate(name: string)
    - Constructor representing a valid Candidate.
- getName(): string
    - Returns the name of the Candidate. Returns “NULL” if the candidate is not valid.

- getWinner(): bool
    - Returns the win status of this candidate.
- setWinner(w: bool):
    - Sets the win status of this candidate.
- isValid: bool
    - Returns whether the candidate represents a null Candidate.
- addBallot(): int
    - Adds 1 to this candidate's ballot count, then returns the current ballot count.

5.4 Election

- runElection() = 0: void
    - Pure virtual. Validates the ballots, then runs the election algorithm for the specific election type.
- validateBallots() = 0: bool
    - Pure virtual. Validates the ballots in accordance with the election type. Returns whether or not all of the ballots were valid, returns false otherwise.

5.4.1 STV Election

- calculateDroop(numCandidates : int, numBallots : int): double
    - Returns the droop quota amount based on the number of candidates and number of ballots.
- shuffleBallots(): void
    - Shuffles the order of the entire ballot vector. To be done before the STV algorithm is run.
- doRound(): bool
    - Runs through one round of the STV algorithm. Iterating through the ballots puts them into the usedBallotList, adding to their representative candidate. If the candidate reaches the group quota, then the ballots in the usedBallotList representing the candidate are put into removedBallotList and the method returns true. If there are no more ballots in the main ballot list, returns false.
- distributeBallots(cand: Candidate): void
    - Removes the candidate from each ballot in the used ballot list, and each ballot that returns true on this removal and is not a null vote is re-added to the formerly empty main ballot list.

5.4.2 Plurality Election

- doRound(): void
    - Runs through one round of the Plurality algorithm. Iterates through the ballots and puts them into the usedBallotList adding to their representative candidate. Finishes on final ballot.

6. HUMAN INTERFACE DESIGN

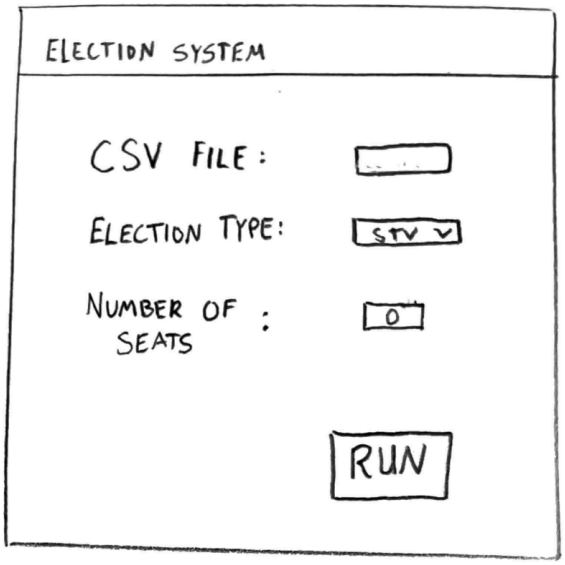
6.1 Overview of User Interface

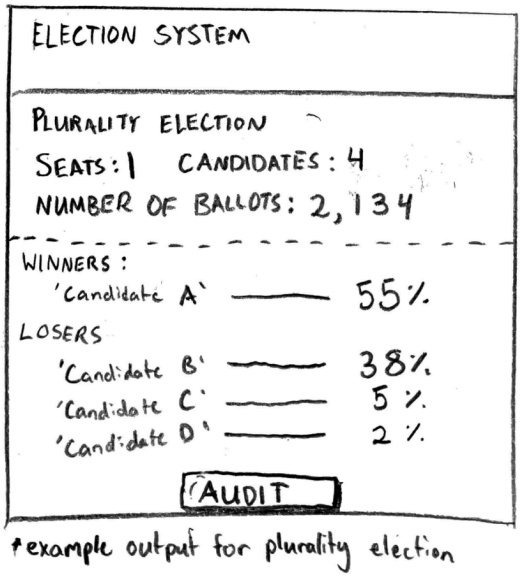
Before the system starts, testers enter a flag to the command line to toggle shuffle mode for STV elections.

On startup, the user interacts with the system through the GUI. The user is prompted to enter the election type, CSV file, and number of seats. Once the user selects the 'Run' button, the Election System processes the ballots and candidates to determine the winners.

Once the election has been processed, the election results are displayed on the GUI based on the type of election. The user will also be prompted to create an audit file for the election results via the 'Audit' button.

6.2 Screen Images





*example output for plurality election

6.3 Screen Objects and Actions

The ‘CSV File’ row gives the user an input field for the CSV file name. It is assumed that the CSV is in the project directory on startup. The ‘Election Type’ row gives the user a dropdown list of the election types (STV, plurality). The ‘Number of Seats’ row provides an input field for the number of seats for the given election.

The ‘RUN’ button acts as the user’s submission button. It begins the election process assuming that all data is valid.

Once the election has been processed, the election results will be displayed on the GUI. The ‘AUDIT’ button can be selected to generate an audit file containing information concerning the results.

7. Requirements Matrix

UC_001	The displayPrompt() function in the UserInterface class handles prompting the user through the GUI.
UC_002	The **sendFile** function in the **UserInterface** class is responsible for bringing the file into the system so that it can be processed.
UC_003	The **createElection** function in the **UserInterface** class is responsible for taking information from the CSV and processing it into ballot and candidate objects.
UC_004	The **runButtonPressed** function in the **UserInterface** class is responsible for handling the user pressing ‘RUN’. This initiates the createElection() function
UC_005	The **runElection** function in the **STVElection** class is responsible for running the STV algorithm given the ballot and candidate information from the CSV file.
UC_006	The **runElection** function in the **PluralityElection** class is responsible for running the plurality algorithm given the ballot and candidate information from the CSV file.
UC_007	The **calculateDroop** function in the **STVElection** class is responsible for calculating the droop quota given the number of candidates and number of ballots.
UC_008	The **generateResults** function in the **UserInterface** class is responsible for displaying the results to the GUI. This function is called from the **Election** class when the processing has concluded
UC_009	The **generateAudit** function in the **UserInterface** class handles when a user presses the ‘AUDIT’ button. This function calls the **generateAudit** function in the **Election**

	class, where the file is created.
UC_010	The shuffle flag is determined on program startup through a command line argument. The boolean **shuffle** in the **Election** class stores whether ‘shuffle’ has been toggled.

8. APPENDICES

Appendix A - GUI Implementation

The GUI for the Election System will be implemented with the open-source software ‘Qt’. Documentation for this software can be found at https://doc.qt.io/.

Software Requirements Specification

for

Election System

Version 1.0 approved

Prepared by Daniel Ginzburg, Austin Odusanya,

Samuel Thorson, Isaac Wodele

University of Minnesota - Team 8

02/21/25

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Election System	02/21	First submission	1.0

1. Introduction

1.1 Purpose

This **software requirements specification (SRS)** document displays a comprehensive description of the automated election software system. The functionality is a new, standalone system that counts ballots for single transferable vote (**STV**) and plurality elections. It explains the purpose of the system, the features of the system, the interfaces of the system, how it should function, as well as the functional and non-functional requirements of the system. This document is meant for users and testers of the system as well as fellow developers.

1.2 Document Conventions

This document follows the conventions laid out by the IEEE System Requirement Specification document.

1.3 Intended Audience and Reading Suggestions

The intended audience of this document is any elected official or system tester who is going to be directly interacting with the system. This includes election officials who are specifically interested in automated election processing via **CSV** files and software system testers who want to gain a deeper understanding of the system. This document will also provide further information for software developers implementing the system.

1.4 Product Scope

Election officials use the Election System to determine election results from a **CSV** file. By providing the file name, number of seats in the election, and the type of election, election officials can use the system to calculate the winners. The system supports two types of elections: **STV** and plurality. This is determined at run time by the election official. After the election algorithm has finished, the results are displayed via **GUI**.

1.5 References

IEEE Computer Society, "IEEE Recommended Practice for Software Requirements Specifications," IEEE Std 830-1998, 1998.

Qt Wiki, “Qt for Beginners,” February 21, 2025, https://wiki.qt.io/Qt_for_Beginners

Canvas, “Gephi SRS,” February 2017

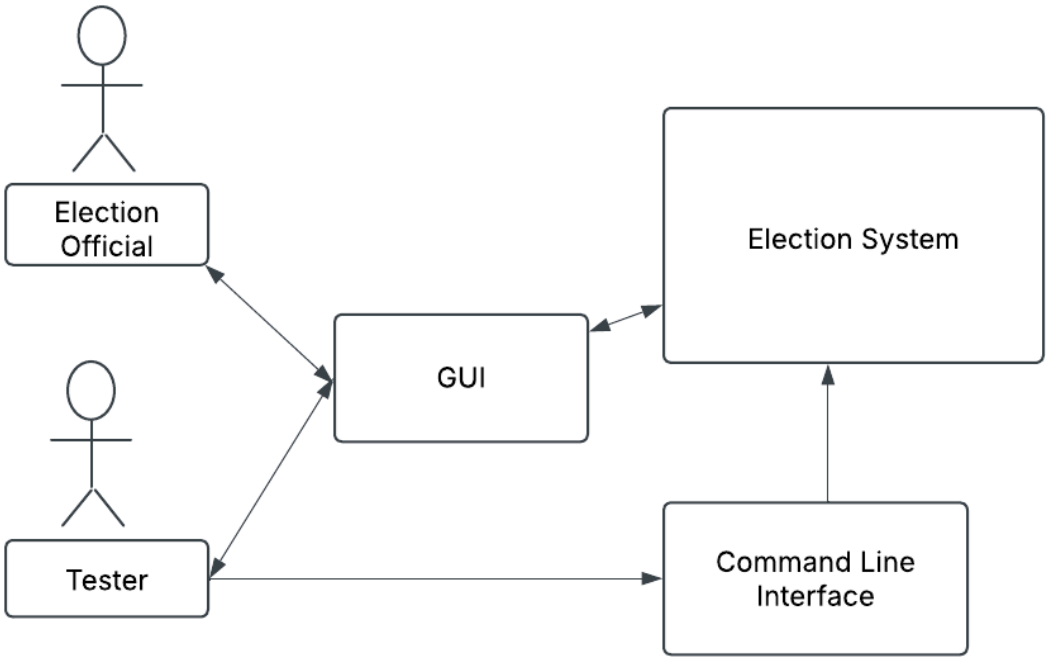
Canvas, “Webapp SRS,” April 15, 2004

Canvas, “Voting System Description,” February 21, 2025

2. Overall Description

2.1 Product Perspective

This product is a standalone system made to aid election officials in calculating election results. It is used by both election officials and testers in a local environment.



Both the election officials and testers access the program through the **GUI**. The tester also accesses the program directly through the command line interface (**CLI**).

2.2 Product Functions

2.2.1 Functions for Election Officials and System Testers

Use Case: **Input Election Information**

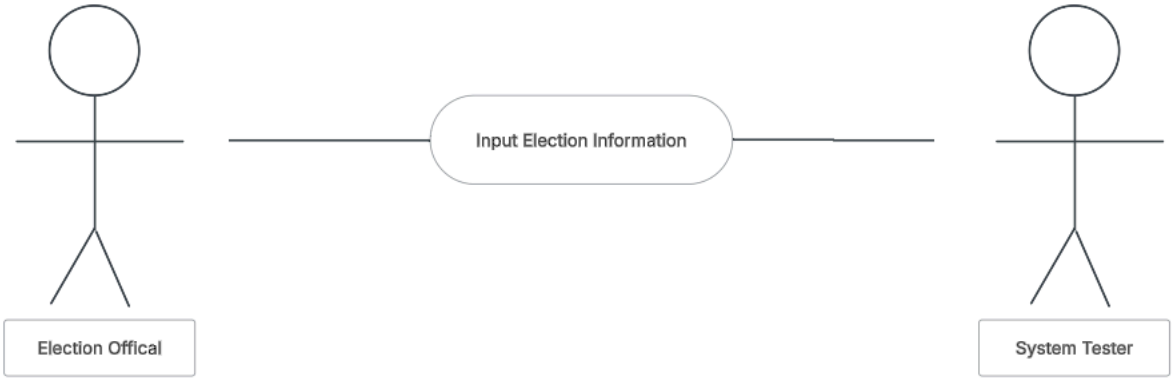


Figure 1: The election official and system tester can input election information.

Step by Step:

1. The user has started the system.
2. The user is prompted to input the election information by the **GUI**.
3. The user successfully inputs the correct information for the election.

ref: Section 4.1, Prompt User Information

Use Case: **Run Election (STV or Plurality)**

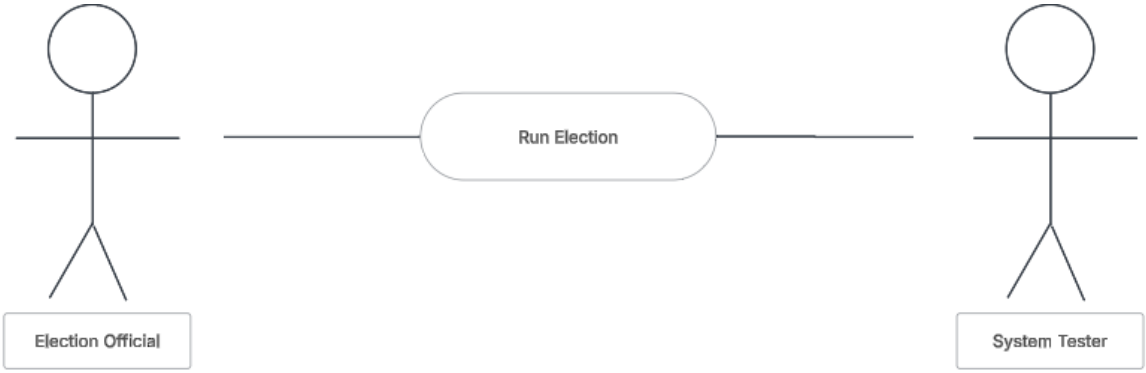


Figure 3: The election official and system tester can run the election

Step by Step:

1. The user has entered valid election information.
2. The user selects to run the election via the **GUI**.
3. The system begins running the election.

ref: Section 4.4, Run Election

Use Case: **View Results**

Figure 4: The election official and system tester can view results via the **GUI**.

Step by Step:

1. The system displays the results to the **GUI**.
2. The user views the results.

ref: Section 4.8, Display Results To **1**

Use Case: **Request Audit**

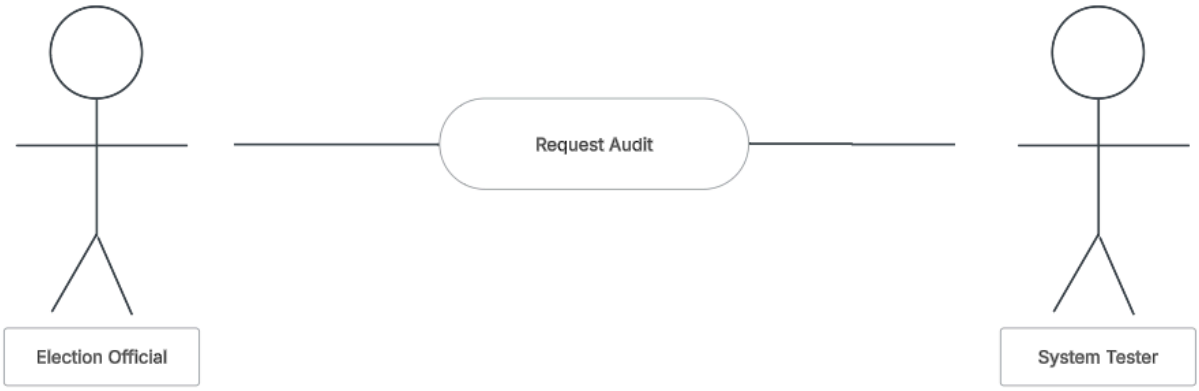


Figure 5: The election official and system tester can request an audit file.

Step by Step:

1. The user requests an audit file of the results.
2. The system provides an audit file containing the election results.
3. The user views the audit file.

ref: Section 4.9, Creating Audit File

2.2.1 Functions for only System Testers

Use Case: **Interact with Command Line**

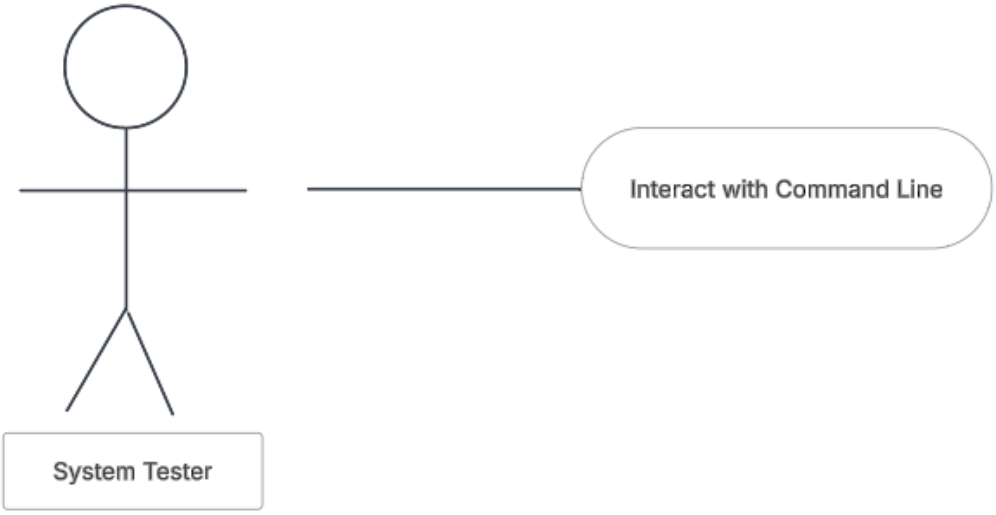


Figure 6: The system tester can interact with the Command Line.

Step by Step:

1. The system starts up.
2. The system tester interacts with the command line.

ref: Section 4.10, Toggle Shuffle in **CLI**

Use Case: **Toggle Shuffle**

Figure 7: The system tester can toggle shuffle.

Step by Step:

1. The system tester interacts in the command line.
2. The system tester enables/disables shuffle via a command line flag.

ref: Section 4.10, Toggle Shuffle in **CLI**

Use Case: **Run Test File**

Figure 8: The system tester can run a test file.

Step by Step:

1. The system tester enters valid test election information.
2. The system runs a test election.

ref: Section 4.4, Run Election

Use Case: **View Results**

Figure 9: The system tester is able to run a test file.

Step by Step:

3. The system runs a test election.
4. The system displays the test results to the **GUI**.
5. The system tester views the test results.

ref: Section 4.8, Display Results to **GUI**

2.3 User Classes and Characteristics

Two roles have access to the system: the tester and the election officials. Both roles can perform the same functions outside of the tester. The tester can toggle the shuffle option in the client when running the program.

2.4 Operating Environment

- Windows 2000
- Windows XP
- Windows Vista
- Windows 7
- Windows 8
- Windows 10
- Mac OS X
- Linux

2.5 Design and Implementation Constraints

The Election System is developed in C++. The **GUI** is built with the graphical toolkit Qt.

2.6 User Documentation

There is no user documentation delivered alongside the software.

2.7 Assumptions and Dependencies

The voting is handled separately from the system developed. Ballots are cast online and delivered in the form of a **CSV**. There will be no mistakes in the files. Security concerns are handled at the voting precincts.

3. External Interface Requirements

3.1 User Interfaces

Information will be displayed to the user as text or **GUI** throughout the program. The user will be prompted for inputs at the beginning of the program through text and the command line. The only other display will occur at the end of the program when it shows a small summary of the election using a **GUI**. Testing the program will allow the user to input a flag to toggle the shuffling of ballots during the initial prompting phase.

3.2 Hardware Interfaces

The Election System must be able to run on a CSE Lab Machine. A system with these specifications can handle no more than 100,000 ballots in under 5 minutes.

3.3 Software Interfaces

The Election System requires GNU Compiler Collection (**GCC**) or another C++ compiler to be installed on the local system. Additional information can be found in section 2.7 of this document.

3.4 Communications Interfaces

The Election System does not require any communication utilities. All capabilities can be executed locally.

4. System Features

4.1 Prompt User Information

Name	Prompt User Information
ID	UC_001
Description	The **GUI** prompts the user for a filename to be entered, the number of seats, and the type of election.

Actors	Election official, tester
Organization Benefits	Allows the user to input information to run an election on specified data.
Frequency of Use	Prompt User Information is used every time the program is run.
Triggers	On startup of the program.
Preconditions	The program is running.
Postconditions	The program has a valid file name for ballot data. The program has a valid election type. The program has a valid number of seats.
Main Course	1. The user is prompted to enter the filename, the number of seats, and the type of the election. 2. The user enters the name of the file. 3. The user enters the number of seats. 4. The user enters the type of election to run.
Alt Courses	None.
Exceptions	EX1: User enters an invalid file name. 1. Tell the user the file name is invalid. 2. Return to Main Course step 1. EX2: User enters an invalid number of seats. 3. Tell the user the number of seats is invalid. 4. Return to Main Course step 1. EX3: User enters an invalid election type. 5. Tell the user the election type is invalid. 6. Return to Main Course step 1.

4.2 Bring File into System

Name	Bring File Into System
ID	UC_002
Description	After the user enters the file name, the file is opened and ready to be processed by the system.
Actors	Election official, tester

Organizational Benefits	Loading in the file allows the system to process it.
Frequency of Use	Every time the user enters a file name, the system will load the file.
Triggers	The user enters the file that is to be loaded.
Preconditions	The user enters the file into the system.
Postconditions	The file is opened and ready to be processed.
Main Course	1. The system attempts to open the file with the given file name. 2. The system successfully opens the file (see EX1). 3. The file is open and ready to be processed.
Alt Courses	None.
Exceptions	EX1: File failed to open. 1. Notify the user that the file failed to open. 2. See UC_001.

4.3 Process CSV file

Name	Process CSV File
ID	UC_003
Description	The file is read from the file handler and processed into candidates and ballots.
Actors	Election official, tester
Organizational Benefits	Analyze and determine if the file is in proper election format so it can be used by the election algorithm.
Frequency of Use	Every time a file is successfully opened by the program.
Triggers	Every time a file is successfully opened by the program.
Preconditions	The file handler opened a valid file (See UC_002).
Postconditions	The file has been processed into candidates and ballots for the election.

Main Course	1. The file is read from the open file handler. 2. The format of the file is compared to the type of election entered by the user (see EX1). 3. The file is processed into a data structure containing ballots and candidate information.
Alt Courses	None.
Exceptions	EX1 File format doesn't match the election type. 1. Notify the user that the file's format doesn't match the chosen election type. 2. Reprompt for user information (see "Prompt User Information").

4.4 Run Election

Name	Run Election
ID	UC_004
Description	Election process is begun once the user presses 'Run Election' in the GUI.
Actors	Election official, tester
Organizational Benefits	Allows the election process to begin.
Frequency of Use	Every time the user runs the program with election information.
Triggers	The user presses the 'Run Election' button in the GUI.
Preconditions	1. The user inputted a valid **CSV** file. 2. The user selected the type of election (**STV**, plurality).
Postconditions	1. The program begins running the election with the inputted parameters.
Main Course	1. The user selects the button 'Run Election'. 2. The program picks an algorithm to run. 3. The program begins processing the **CSV** file. (See UC_003)
Alt Courses	None.

Exceptions	EX1: The user does not select an election type.

4.5 Running STV algorithm

Name	Running STV Algorithm
ID	UC_005
Description	After the file has been successfully loaded, and the user has clicked start on the GUI.
Actors	Election official, tester
Organizational Benefits	Creates an automated testing system that reduces human error and produces an accurate result.
Frequency of Use	Whenever the user enters **STV** as the election type or the tester runs a program from **CLI**.
Triggers	Start is clicked on the **GUI** with a valid ballot data structure, or function called from **CLI** from a tester.
Preconditions	See PRE_001.
Postconditions	STV results are displayed to the user.
Main Course	1. The system determines whether or not the system was run from the CLI (see AC1, AC2). 2. Droop Quota is calculated (See UC_006). 3. Check that the total ballot count is nonzero (see EX1). 4. Look at the next ballot (see AC3, AC4). 5. Open **GUI** if not open and display election results on **GUI**.
Alt Courses	AC1: System run from **GUI**. 1. Shuffle the ballots. 2. Return to Main Course Step 2. AC2: System run from **CLI**. 1. See UC_002. 2. See UC_003. 3. Shuffle ballots if argument omitted, otherwise only shuffle it if tester desires. 4. Return to Main Course Step 2. AC3: Next Ballot selected 1. Based on ranking within the ballot and nominees left,

	determine who the ballot would count for (see AC5, AC6). AC4: No more Ballots. 1. Set the ballot list to the fewest voted nominee ballot list. 2. Remove the nominee from the election. 3. Return to Main Course Step 4. AC5: Ballot supports a nominee. 1. Add ballot to nominee ballots. 2. If the Droop quota is not reached, return to Main Course step 4. 3. Add nominee to list of winners. 4. Remove nominee ballots from the ballot list. 5. If all seats are filled, return to Main Course step 5. 6. Return to Main Course step 4. AC6: Ballot fails to support a nominee. 1. Remove the ballot from the list of ballots. 2. Return to Main Course step 4.
Exceptions	EX1 Ballot Count is 0 1. **GUI** opens if not open and displays “No Winners”.

Precondition ID	User	Tester
PRE_001	**GUI** delivers a valid file to the program, the file gets processed successfully (UC_003), and a valid data structure with the ballet information is passed. **STV** is the election type on the **GUI**.	**STV** is chosen on the command line. A filename is provided. Optionally a shuffle argument is included which defaults to true.

4.6 Running Plurality algorithm

Name	Running Plurality algorithm
ID	
Description	The plurality algorithm will run and determine the winners of the election.

Actors	Election official, tester
Organizational Benefits	Creates an automated testing system that reduces human error and produces an accurate result.
Frequency of Use	Once per election.
Triggers	The CSV file has been properly processed.
Preconditions	The ballots are fully processed and populated.
Postconditions	The result of the election is determined.
Main Course	1. Ballots are assigned and totaled to a candidate. 2. The candidates with the most votes are declared the winners. 3. Tie breakers are broken through random selection.
Alt Courses	None.
Exceptions	None.

4.7 Running Droop Quota algorithm

Name	Running Droop Quota algorithm
ID	UC_007
Description	The Droop Quota algorithm is run to process the ballots and determine the list of elected and non-elected officials.
Actors	Election official, tester
Organizational Benefits	Reduces the number of iterations the **STV** algorithm has to run, saving time.
Frequency of Use	The Droop Quota algorithm is run every time the election program is run.
Triggers	The **STV** algorithm is run.
Preconditions	The **CSV** has processed ballots. The **STV** algorithm is being run.
Postconditions	Generated lists of elected and non-elected officials.

Main Course	1. The number of ballots and number of seats are used to calculate Droop Quota.
Alt Courses	None.
Exceptions	EX 1: The number of ballots is invalid. 1. The system displays the number of ballots is invalid. 2. System terminates. EX 2: The number of seats is invalid. 1. The system displays the number of ballots is invalid 2. System terminates.

4.8 Display Results to GUI

Name	Display Results to GUI
ID	UC_008
Description	Display the type of election, number of ballots, seats, and candidates to the screen. This will also include a list of the winners and losers.
Actors	Election official, tester
Organizational Benefits	Collects information and displays it in an organized fashion.
Frequency of Use	Results are displayed once each time an election is run in the system.
Triggers	Once the chosen algorithm is completed and results have been compiled.
Preconditions	The election has ended and the winners have been decided.
Postconditions	A snapshot of the election results are displayed on the screen.
Main Course	1. A text display will appear in the console that lists all the information mentioned above.
Alt Courses	AC1: If the election is STV, display winners in ascending order. AC2: If the election is plurality, display winners by percentage of votes.

Exceptions	None.

4.9 Creating audit file

Name	Creating audit file
ID	UC_009
Description	The program creates an audit file when requested by the user.
Actors	Election official, tester
Organizational Benefits	Summarizes and compacts election results into one readable file.
Frequency of Use	Runs when the user requests an audit after completion of the election program.
Triggers	The user requests an audit file and inputs a valid file name.
Preconditions	1. The user requests a file with a valid file name. 2. Ballots have been processed.
Postconditions	1. An audit file is created with a file name selected by the user.
Main Course	1. The program receives requests for audit files with file name 2. The program creates a file containing ballot information.
Alt Courses	AC1: The user opts not to create an audit file.
Exceptions	EX:1 No available space for a file. 1. The audit file is not created. 2. GUI announces that there is not enough available space.

4.10 Toggle shuffle in CLI

Name	Toggle shuffle in **CLI**
ID	UC_010
Description	Allows the user to input command line argument to toggle the

	shuffle command.
Actors	Tester
Organizational Benefits	Provides the tester with a more efficient way to verify the **STV** algorithm.
Frequency of Use	Every time the program is run from the command line.
Triggers	When the tester passes a command line argument to the program in the **CLI**.
Preconditions	The command line argument is found.
Postconditions	Shuffle mode is set based on the command line argument.
Main Course	1. The command line input is read. 2. The shuffle flag is read. 3. Shuffle mode turned off.
Alt Courses	1. No command line argument was found.
Exceptions	EX1: Command line argument not valid. 1. The command line prints a statement saying it's not valid. 2. Return to MC Step 1.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

An election must be processed in under 5 minutes with no more than 100,000 ballots.

5.2 Safety Requirements

The Election System does not have any safety requirements.

5.3 Security Requirements

The Election System does not have any security requirements.

5.4 Software Quality Attributes

The Election System is created for reliability, accuracy, and usability. It supports plurality and single transferable voting (**STV**) methods, ensuring flexibility for different election types. Testability is improved through client terminal flags, allowing for a controlled testing environment. Correctness is maintained through the droop quota, while robustness is added by removing incomplete **STV** ballots. Usability is a strength as the system uses simple prompts and **GUI** options to relay information to the users. Overall, the system efficiently determines the outcomes of elections.

5.5 Business Rules

Two roles have access to the system: the tester and the election officials. Both roles can perform the same functions outside of the tester. The tester can toggle the shuffle option in the client when running the program.

6. Other Requirements

Appendix A: Glossary

- **GUI**: Graphical user interface
- **CLI**: Command line interface
- **STV**: Single transferable vote
- **SRS**: Software requirements specification
- **GCC**: GNU Compiler Collection
- **CSV**: Comma-separated values

Appendix B: Analysis Models

Droop Quota formula:

$$[(numberOfBallots \div numberOfSeats)] + 1$$

Appendix C: To Be Determined List

There are no ‘to be determined’ references.