# Team 11
# Voting System
# Software Design Document

Names: Hoai Bui, Ryan Mower, Eric Palmer, Emma Spindler
Date: 02/28/2021

# Table of Contents

# 1. Introduction

## 1.1 Purpose

This software design document describes the architecture and system design of the Voting System for Open Party elections and Instant Run-Off elections to determine a winner for the type of election. This document is made for the developers and testers of the Voting System.

## 1.2 Scope

The Voting System is a tool intended to be used for both Instant Runoff and Open Party Listing elections. Election officials will organize ballots into a CSV file and the software will perform the appropriate operations in order to determine the winner(s) based on the specified election type. The results along with other important information regarding the election will be returned in an audit and media report to be reviewed and shared. The elected officials will be able to run the Voting System to get results from the election and will be provided a media file to share with media personnel. The Voting System allows up to 100,000 ballots to be read in 8 minutes allowing the winner to be determined in a more timely manner. The Voting System eliminates any human error in counting ballots assuming the CSV file is accurate.

## 1.3 Overview

Section 2 contains the project design and the functionality provided by the program to the user. Section 3 consists of the program structure and the various subsystems that make up the overall system of the project. Section 4 explains how data will be stored and organized, as well as the major data types used in the project. Section 5 describes a more detailed look at the program, explaining the algorithms and function implementation. Section 6 explains how the user will interact with the project by detailing the user interface and the user's methods of inputting and receiving data. Section 7 discusses how our project meets all the requirements detailed in the SRS.

## 1.4 Reference Material

Sources of information for this document were the SDD template and example, assignment description from Canvas, as well as the SRS Document PDF which explained how the voting system works.

## 1.5 Definitions and Acronyms

- **ALBF** - Accept and load a CSV ballot file.

- **Ballot file** - The ballot file is the comma separated file that contains user's votes. This file will then be passed to the Voting System algorithm to calculate the results of the election.
- **CSV** - Comma separated value file. This is a file that contains data that is delimited by a comma.
- **Election information** - A vague term that represents election statistics. It can be broken down into three-sub categories denoted by the context in which the phrase is used.
    - Audit file election information - How the election progressed (which vote went to which candidate), election type, number of seats and ballots, candidates and their party, winners and losers, and which candidates were eliminated at specific rounds.
    - Terminal/Media report election information - The results and general information about the election.
    - Voting information - Who the people voted for, candidates and their party, election type, number of ballots, and the number of seats. This information will compose the ballot CSV file, so everything necessary to compute election results must be inside this file.
- **MISC** - Miscellaneous
- **OPL** - Open Party List. It is a style of voting where candidates are elected based on the number of total votes and the number of seats to be filled.
- **PER** - Produce and distribute reports displaying election statistics.
- **IR -** Instant Runoff. It is a style of voting where the candidate with the fewest number of votes is continually eliminated, with their votes going to the next highest choice marked on the ballot until a candidate has the majority of votes, or popularity if two candidates remain.
- **SDD** - Software design document. It is this document, which describes how the software requirements are met with data structures, interfaces, and functions.

## 2. System Overview

Our project is a voting system which can process a CSV file of votes to produce statistics about the election in the form of a media or audit file, as well as displaying these results to the terminal. It will also have the ability to recreate each step of the voting process along the way. It is made up of five different classes which contribute to the overall functionality of the program.

1. The Ballot class which stores and updates information about each ballot.

2. The Candidate class which stores and updates information about each candidate.
3. The Driver which reads information from the CSV file into the program.
4. The Election class which computes election results.
5. The Report class which creates reports and outputs election results to the screen.

## 3. System Architecture

### 3.1. Architectural Design

There are three main subsystems within the Voting System program.

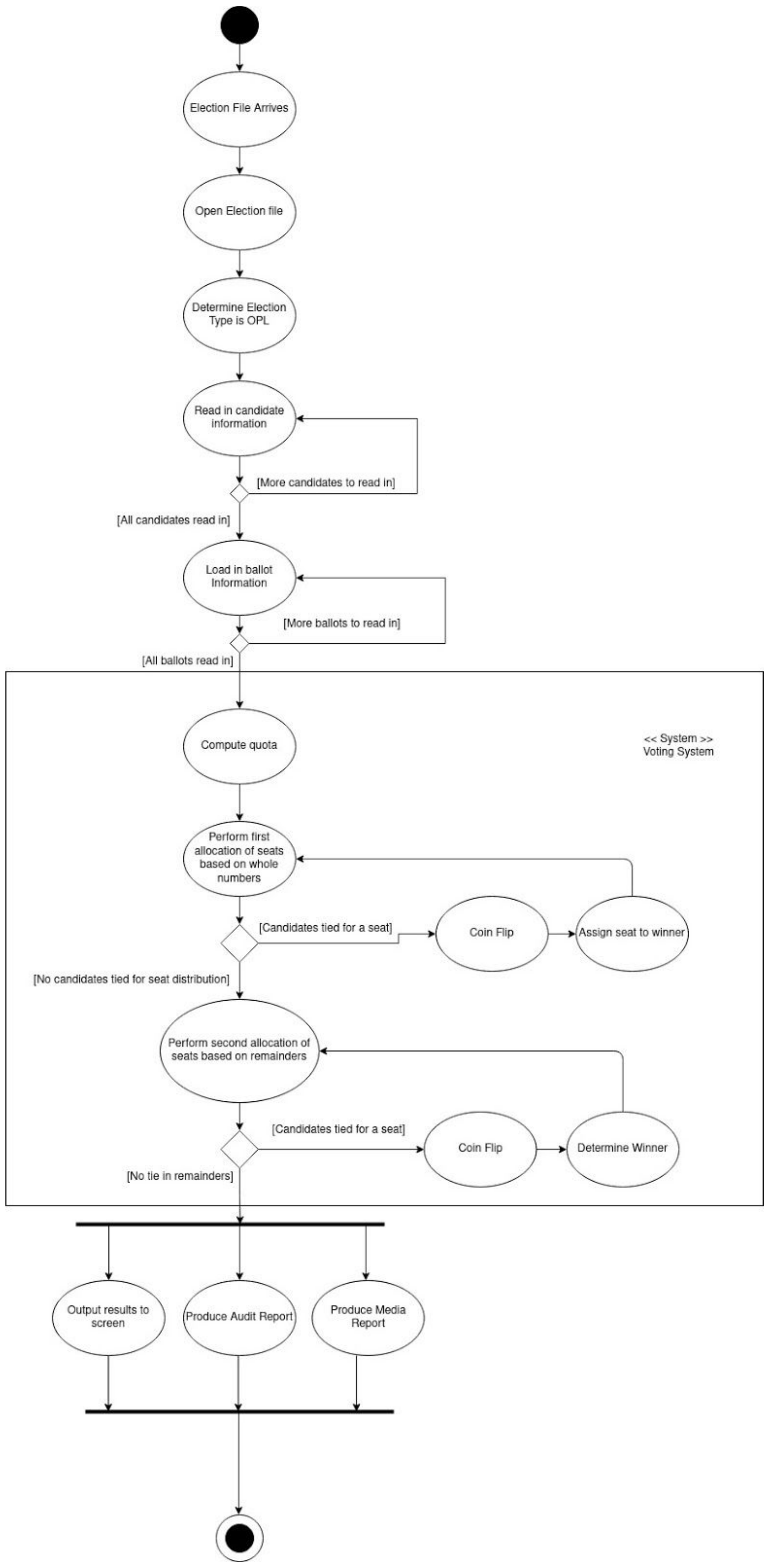
1. Subsystem 1
    a. Loading Election Information from a CSV file.
2. Subsystem 2
    a. The two different election algorithms, Open Party Listing and Instant Runoff.
3. Subsystem 3
    a. Generation of election results, outputting the data to the screen and the creation of the media and audit report.

(Subsystem 1) The first subsystem, the loading of election information from a CSV file, is the first module activated during program execution. This subsystem is responsible for loading all of the election information regarding ballots, number of seats, the candidates and all other required information to compute the election results into memory. It does this by parsing the CSV and assigning the variables accordingly. This will allow for the future substems to perform the appropriate calculations. Once the election information has been loaded into memory, these subsystems' responsibilities are completed and control will be passed down to the next subsystem.

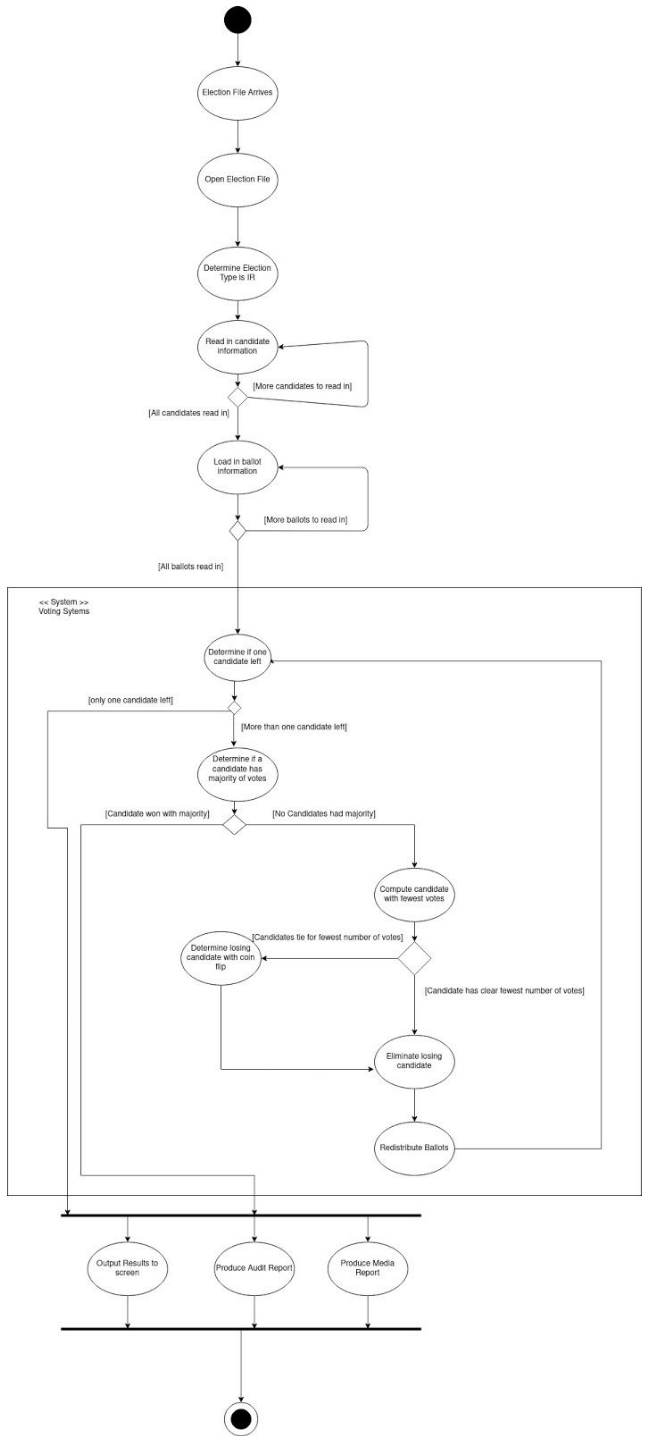
(Subsystem 2) Open Party Listing and Instant Runoff are both election algorithms that are considered subsystems. Their responsibilities include calculating the election results corresponding to the running election type. Since all of the required election information is loaded into memory from the first subsystem, the OPL and IR algorithms will calculate the election results accordingly. After the results are computed, control will be passed onto the final subsystem.

(Subsystem 3) Generation of election results is responsible for outputting the election results in several different forms. These forms of communication include outputting the results to the screen and creating a media and audit report. Once control is directed at this subsystem, it will take the results computed from Subsystem 2 and output them to the screen and create an audit and media report. Both OPL and IR have their own desired output format and Subsystem 3 will abide by these rules.

OPL_Activity_Diagram (Located in SDD folder)-
This diagram shows the flow the Voting System will endure during program execution for an Open Party Listing election type. It depicts the control paths that will be taken depending on the election information.

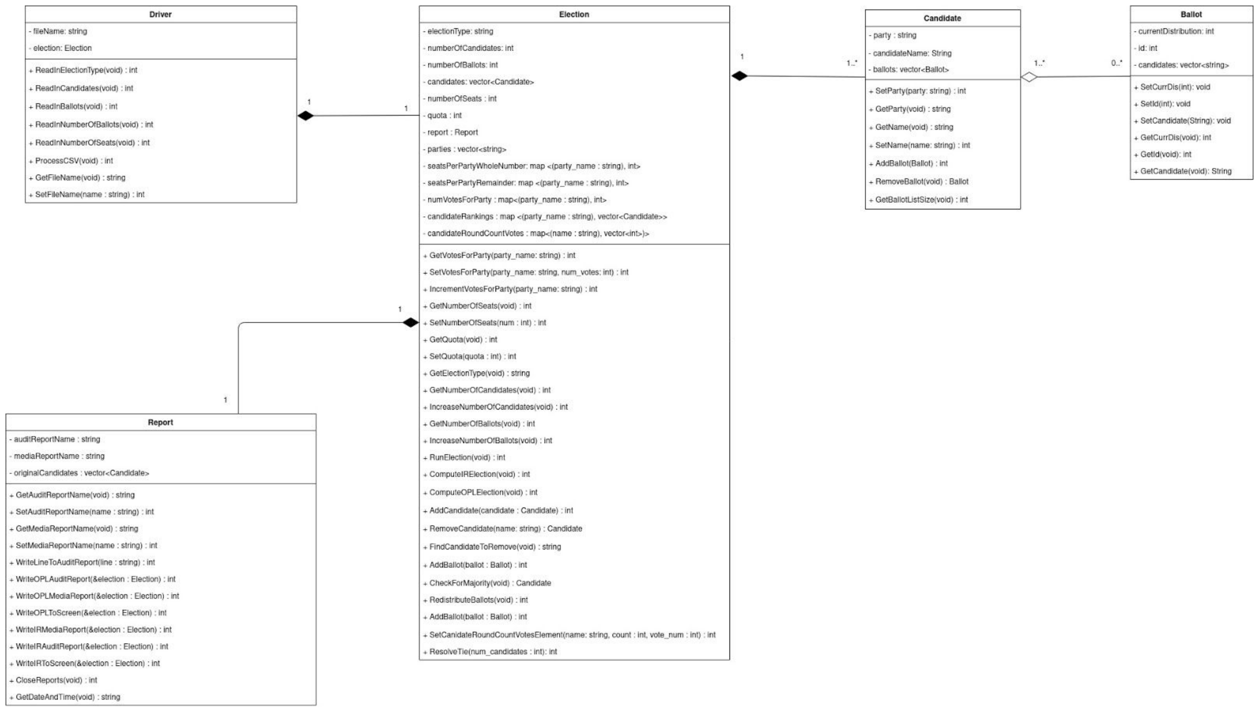

Election File Arrives
Open Election file
Determine Election Type is OPL
Read in candidate information
[More candidates to read in]
[All candidates read in]
Load in ballot Information
[More ballots to read in]
[All ballots read in]
Compute quota
<< System >> Voting System
Perform first allocation of seats based on whole numbers
[Candidates tied for a seat]
Coin Flip
Assign seat to winner
[No candidates tied for seat distribution]
Perform second allocation of seats based on remainders
[Candidates tied for a seat]
Coin Flip
Determine Winner
[No tie in remainders]
Output results to screen
Produce Audit Report
Produce Media Report

IR_Activity_Diagram (Located in SDD folder)-
This diagram shows the flow the Voting System will endure during program execution for an Instant Runoff election type. It depicts the control paths that will be taken depending on the election information.


Election File Arrives
Open Election File
Determine Election Type is IR
Read in candidate information
[More candidates to read in]
[All candidates read in]
Load in ballot information
[More ballots to read in]
[All ballots read in]
<< System >>
Voting Sytems
Determine if one candidate left
[only one candidate left]
[More than one candidate left]
Determine if a candidate has majority of votes
[Candidate won with majority]
[No Candidates had majority]
Compute candidate with fewest votes
[Candidates tie for fewest number of votes]
Determine losing candidate with coin flip
[Candidate has clear fewest number of votes]
Eliminate losing candidate
Redistribute Ballots
Output Results to screen
Produce Audit Report
Produce Media Report

## 3.2 Decomposition Description

UML_Class_Diagram (Located in SDD folder)-
The UML class diagram displays the class layout, structure, and interactions for the Voting System.


Driver
- fileName: string
- election: Election
+ ReadInElectionType(void) : int
+ ReadInCandidates(void) : int
+ ReadInBallots(void) : int
+ ReadInNumberOfBallots(void) : int
+ ReadInNumberOfSeats(void) : int
+ ProcessCSV(void) : int
+ GetFileName(void) : string
+ SetFileName(name : string) : int
1
1
Election
- electionType: string
- numberOfCandidates: int
- numberOfBallots: int
- candidates: vector<Candidate>
- numberOfSeats : int
- quota : int
- report : Report
- parties : vector<string>
- seatsPerPartyWholeNumber: map <(party_name : string), int>
- seatsPerPartyRemainder: map <(party_name : string), int>
- numVotesForParty : map<(party_name : string), int>
- candidateRankings : map <(party_name : string), vector<Candidate>>
- candidateRoundCountVotes : map<(name : string), vector<int>)>
+ GetVotesForParty(party_name: string) : int
+ SetVotesForParty(party_name: string, num_votes: int) : int
+ IncrementVotesForParty(party_name: string) : int
+ GetNumberOfSeats(void) : int
+ SetNumberOfSeats(num : int) : int
+ GetQuota(void) : int
+ SetQuota(quota : int) : int
+ GetElectionType(void) : string
+ GetNumberOfCandidates(void) : int
+ IncreaseNumberOfCandidates(void) : int
+ GetNumberOfBallots(void) : int
+ IncreaseNumberOfBallots(void) : int
+ RunElection(void) : int
+ ComputeIRElection(void) : int
+ ComputeOPLElection(void) : int
+ AddCandidate(candidate : Candidate) : int
+ RemoveCandidate(name: string) : Candidate
+ FindCandidateToRemove(void) : string
+ AddBallot(ballot : Ballot) : int
+ CheckForMajority(void) : Candidate
+ RedistributeBallots(void) : int
+ AddBallot(ballot : Ballot) : int
+ SetCanidateRoundCountVotesElement(name: string, count : int, vote_num : int) : int
+ ResolveTie(num_candidates : int): int
1
1..*
Candidate
- party : string
- candidateName: String
- ballots: vector<Ballot>
+ SetParty(party: string) : int
+ GetParty(void) : string
+ GetName(void) : string
+ SetName(name: string) : int
+ AddBallot(Ballot) : int
+ RemoveBallot(void) : Ballot
+ GetBallotListSize(void) : int
1..*
0..*
Ballot
- currentDistribution: int
- id: int
- candidates: vector<string>
+ SetCurrDis(int): void
+ SetId(int): void
+ SetCandidate(String): void
+ GetCurrDis(void): int
+ GetId(void): int
+ GetCandidate(void): String
1
1
Report
- auditReportName : string
- mediaReportName : string
- originalCandidates : vector<Candidate>
+ GetAuditReportName(void) : string
+ SetAuditReportName(name : string) : int
+ GetMediaReportName(void) : string
+ SetMediaReportName(name : string) : int
+ WriteLineToAuditReport(line : string) : int
+ WriteOPLAuditReport(&election : Election) : int
+ WriteOPLMediaReport(&election : Election) : int
+ WriteOPLToScreen(&election : Election) : int
+ WriteIRMediaReport(&election : Election) : int
+ WriteIRAuditReport(&election : Election) : int
+ WriteIRToScreen(&election : Election) : int
+ CloseReports(void) : int
+ GetDateAndTime(void) : string

The above diagram depicts the class layout and structure for the Voting System. Driver is the class that drives the Voting System program. It loads all of the data into computer memory, so the Election class can properly compute the election results. Control is transferred from Driver to Election. Here, the Election class will call ComputeOPLElection or ComputeIRElection accordingly. Once the algorithm has terminated, control will be passed to Report. Report will generate the media and audit report, and output the results to the screen. Both Candidate and Ballot classes are used to model the candidates and the votes from the people participating in the election. They are important when assigning ballots to candidates because it shows how many votes each candidate has.

## 3.3 Design Rationale

One main architectural design was necessary for improved program efficiency. If ballots were the only form of tracking which candidate it is for, redistributing them would require constant traversal over the list of ballots to update their candidate. This proves incredibly inefficient and solved by having specific candidates contain a list of ballots that are for them. In the Instant Runoff election, ballots corresponding to an eliminated candidate can be redistributed to the ballot’s next favorite candidate. This will prevent touching all ballots when it is not required.

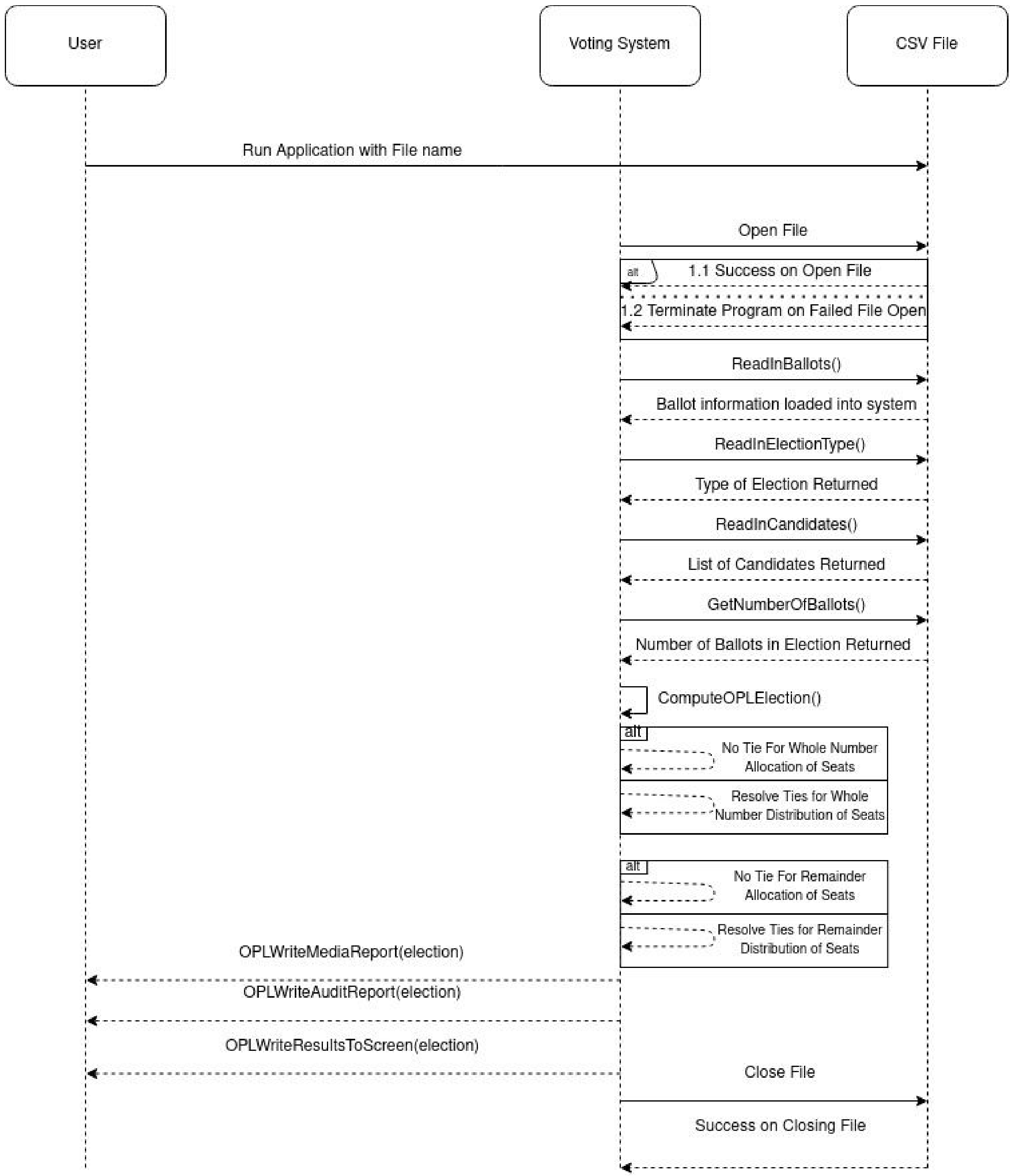
Reporting inside the Election class was also heavily considered. Reporting inside the Election class would allow for dynamic ballot tracking and progression of the election. However, it would be difficult to produce the overarching summary if reporting in the Election class was the only way. To solve this, a combination of a Reporting class and reporting within Election will be used. The ballot tracking will be done in the Election class during Voting System execution. Then, the Report class will output a summary of the results from the election.

Additionally, breaking up the Voting System into three main modules seemed most trivial. This is because the program must first load in all of the election information, then it must perform the appropriate computation, and finally create the appropriate documentation for the election results. Since there are three main sections, breaking up the Voting System architecture into the three modules seemed most fitting.

# 4. Data Design

## 4.1 Data Description

Sequence Diagram (Located in SDD folder) - Below is the sequence diagram which displays the interactions between the various entities involved with the Voting System.


User
Voting System
CSV File
Run Application with File name
Open File
alt
1.1 Success on Open File
1.2 Terminate Program on Failed File Open
ReadInBallots()
Ballot information loaded into system
ReadInElectionType()
Type of Election Returned
ReadInCandidates()
List of Candidates Returned
GetNumberOfBallots()
Number of Ballots in Election Returned
ComputeOPLElection()
alt
No Tie For Whole Number Allocation of Seats
Resolve Ties for Whole Number Distribution of Seats
alt
No Tie For Remainder Allocation of Seats
Resolve Ties for Remainder Distribution of Seats
OPLWriteMediaReport(election)
OPLWriteAuditReport(election)
OPLWriteResultsToScreen(election)
Close File
Success on Closing File

The Voting System starts with a CSV file in which all of the needed information regarding the election is stored. A Driver class parses the CSV file and uses that information to initialize the election. All ballots, candidates, elections, and reports are stored as their own objects within the Voting System. The information from the CSV file is loaded into the Ballot and Candidate classes. From here, the Election class will compute the specified election. Then, the report class is called to output the election results to the terminal as well as in the form of audit and media reports. No databases are used because all the information is stored in memory.

## 4.2 Data Dictionary

| **Class** | **Data Members** | **Methods** |
|---|---|---|
| **Ballot**<br>This class creates an object for every ballot. | - **candidates(vector<string>)**<br>The candidate(s) voted for<br><br>- **currentDistribution(int)**<br>Keeps track of which candidate the ballot is being counted for<br><br>- **id(int)**<br>Ballot ID | + GetCandidate(void): String<br><br>+ GetCurrDis(void): int<br><br>+ GetId(void): int<br><br>+ SetCandidate(String): void<br><br>+ SetCurrDis(int): void<br><br>+ SetId(int): void |
| **Candidate**<br>This class creates an object for every candidate. | - **ballots(vector<Ballot>)**<br>Stores all of the ballots the candidate receives<br><br>- **candidateName(String)**<br>Name of the candidate<br><br>- **party(String)**<br>The party in which the candidate belongs to | + AddBallot(Ballot) : int<br><br>+ GetBallotListSize(void) : int<br><br>+ GetName(void) : string<br><br>+ GetParty(void) : string<br><br>+ RemoveBallot(void) : Ballot<br><br>+ SetName(name: string) : int<br><br>+ SetParty(party: string) : int |
| **Driver**<br>This class is responsible for taking in the CSV file and initializing the election. | - **election: Election**<br>The election class<br><br>- **fileName: string**<br>CSV file name | + ReadInElectionType(void) : int<br><br>+ ReadInCandidates(void) : int<br><br>+ ReadInBallots(void) : int<br><br>+ ReadInNumberOfBallots(void) : int<br><br>+ ReadInNumberOfSeats(void) : int<br><br>+ ProcessCSV(void) : int<br><br>+ GetFileName(void) : string<br><br>+ SetFileName(name : string) : int |

| | | |
|---|---|---|
| **Election**<br>This class computes the election. | - **candidateRankings : map <(party_name : string), vector<Candidate>>**<br>(OPL) Ranks the candidates from highest number of votes to lowest for the given party<br><br>- **candidateRoundCountVotes : map<(name : string), vector<int>)>**<br>(IR) used to keep track a candidate's votes for each count<br><br>- **candidates(vector<Candidate>)**<br>Contains all of the candidates<br><br>- **electionType(String)**<br>The type of election<br><br>- **numberOfBallots(int)**<br>The number ballots in the election<br><br>- **numberOfCandidates(int)**<br>The number of candidates in the election<br><br>- **numberOfSeats : int**<br>(OPL) Number of seats open for election<br><br>- **numVotesForParty : map<(party_name : string), int>**<br>(OPL) The number of votes each party receives<br><br>- **parties : vector<string>**<br>Stores the parties<br><br>- **quota : int**<br>(OPL) Used to fill out the proper number of seats per party<br><br>- **report : Report**<br>The report class<br><br>- **seatsPerPartyRemainder: map <(party_name : string), int>**<br>Tracks the seats available based the remainder from the first allocation of seats<br><br>- **seatsPerPartyWholeNumber: map <(party_name : string), int>**<br>The number of seats each party receives based on the whole number allocation | + AddBallot(ballot : Ballot) : int<br><br>+ AddCandidate(candidate : Candidate) : int<br><br>+ CheckForMajority(void) : Candidate<br><br>+ ComputeIRElection(void) : int<br><br>+ ComputeOPLElection(void) : int<br><br>+ GetElectionType(void) : string<br><br>+ GetNumberOfBallots(void) : int<br><br>+ GetNumberOfCandidates(void) : int<br><br>+ GetNumberOfSeats(void) : int<br><br>+ GetQuota(void) : int<br><br>+ GetVotesForParty(party_name: string) : int<br><br>+ FindCandidateToRemove(void) : string<br><br>+ IncreaseNumberOfBallots(void) : int<br><br>+ IncreaseNumberOfCandidates(void) : int<br><br>+ IncrementVotesForParty(party_name: string) : int<br><br>+ SetCandidateRoundCountVotesElement(name: string, count : int, vote_num : int) : int<br><br>+ SetNumberOfSeats(num : int) : int<br><br>+ SetQuota(quota : int) : int<br><br>+ SetVotesForParty(party_name: string, num_votes: int) : int<br><br>+ ResolveTie(num_candidates : int): int<br><br>+ RedistributeBallots(void) : int<br><br>+ RemoveCandidate(name: string) : Candidate<br><br>+ RunElection(void) : int |
| **Report**<br>This class is responsible for creating media and audit reports for the desired election type as | - **auditReportName : string**<br>Name of the audit report<br><br>- **mediaReportName : string**<br>Name of the media report<br><br>- **originalCandidates :vector<Candidate>**<br>All of the original candidates | + WriteIRAuditReport(election) : int<br><br>+ WriteIRMediaReport(election) : int<br><br>+ WriteIRResultsToScreen(election) : int<br><br>+ WriteOPLAuditReport(election) : int<br><br>+ WriteOPLMediaReport(election) : int |

| | | |
|---|---|---|
| well as outputting the results to the screen. | | + WriteOPLResultsToScreen(election) : int<br><br>+ Report(IR) : void<br><br>+ Report(OPL) : void |

## 5. Component Design

### Ballot

| Function | Description |
|---|---|
| + GetCandidate(void): String | Returns the candidate(s) the ballot votes for |
| + GetCurrDis(void): int | Returns the index into the candidate list representing which candidate the ballot is for. |
| + GetId(void): int | Returns the ballot's ID |
| + SetCandidate(String): void | Stores the candidate that the ballot votes for |
| + SetCurrDis(int): void | Sets the index into the candidate list representing which candidate the ballot is for. |
| + SetId(int): void | Stores the ballot ID |

### Candidate

| | |
|---|---|
| + AddBallot(Ballot) : int | Adds a ballot to the list of ballots for the candidate, returning a 0 for success or 1 for failure. |
| + GetBallotListSize(void) : int | Returns the number of ballots in the candidates ballots vector. |
| + GetName(void) : string | Returns the name of the candidate. |
| + GetParty(void) : string | Returns the party of the candidate(s) |
| + RemoveBallot(void) : Ballot | Removes a ballot from the list of ballots for the candidate and returns it. |
| + SetName(name: string) : int | Sets the name of the candidate to the given string, returning a 0 for success or 1 for failure. |

| | |
|---|---|
| + SetParty(party: string) : int | Sets the party of the candidate to the given string, returning a 0 for success or 1 for failure. |

## Driver

| | |
|---|---|
| + ReadInElectionType(void) : int | Sets the election type based on the data in the CSV file, returning a 0 for success or 1 for failure. |
| + ReadInCandidates(void) : int | Sets the list of candidates based on the data in the CSV file, returning a 0 for success or 1 for failure. |
| + ReadInBallots(void) : int | Sets the list of ballots based on the data in the CSV file, returning a 0 for success or 1 for failure. |
| + ReadInNumberOfBallots(void) : int | Sets the number of ballots based on the data in the CSV file, returning a 0 for success or 1 for failure. |
| + ReadInNumberOfSeats(void) : int | Sets the number of seats based on the data in the CSV file, returning a 0 for success or 1 for failure. |
| + ProcessCSV(void) : int | Calls all of the above ReadIn functions to obtain all of the data from the CSV file, returning a 0 for success or 1 for failure. |
| + GetFileName(void) : string | Returns the filename passed by the user when calling the program. |
| + SetFileName(name : string) : int | Sets the filename of the CSV file to the given string, returning a 0 for success or 1 for failure. |

## Election

| | |
|---|---|
| + AddBallot(ballot : Ballot) : int | Adds a ballot to the election, returning a 0 for success or 1 for failure. |
| + AddCandidate(candidate : Candidate) : int | Adds a candidate to the election, returning a 0 for success or 1 for failure. |
| + CheckForMajority(void) : Candidate | Returns the candidate with the majority of votes, or null if there is no such candidate. |
| + ComputeIRElection(void) : int | Determines the results of the instant runoff |

| | |
|---|---|
| | election, returning a 0 for success or 1 for failure. |
| + ComputeOPLElection(void) : int | Determines the results of the open party list election, returning a 0 for success or 1 for failure. |
| + GetElectionType(void) : string | Returns the election type of the election. |
| + GetNumberOfBallots(void) : int | Returns the number of ballots in the election. |
| + GetNumberOfCandidates(void) : int | Returns the number of candidates in the election. |
| + GetNumberOfSeats(void) : int | Returns the number of seats in the election. |
| + GetQuota(void) : int | Returns the quota of the election. |
| + GetVotesForParty(party_name: string) : int | Returns the number of votes for the given party. |
| + FindCandidateToRemove(void) : string | Returns the name of the candidate with the fewest number of votes. |
| + IncreaseNumberOfBallots(void) : int | Increments the number of ballots in the election by 1, then returns the new number of ballots. |
| + IncreaseNumberOfCandidates(void) : int | Increments the number of candidates in the election by 1, then returns the new number of candidates. |
| + IncrementVotesForParty(party_name: string) : int | Increments the number of votes for the given party by 1, then returns the new number of votes for the party. |
| + SetCandidateRoundCountVotesElement(name: string, count : int, vote_num : int) : int | Sets the number of votes in the given round (vote_num) to the given count for the candidate with the given name, returning a 0 for success or 1 for failure. |
| + SetNumberOfSeats(num : int) : int | Sets the number of seats in the election to the given number, returning a 0 for success or 1 for failure. |
| + SetQuota(quota : int) : int | Sets the quota for the election to the given int, based on the number of seats and votes cased |
| + SetVotesForParty(party_name: string, num_votes: int) : int | Sets the number of votes for the party with the given name to the given int, returning a 0 |

| | |
|---|---|
| | for success or 1 for failure. |
| + ResolveTie(num_candidates : int): int | Returns a random int from 1 to num_candidates inclusive indicating who should win the election, called in case of a tie. |

| | |
|---|---|
| + RedistributeBallots(void) : int | Updates the number of ballots voting for each candidate, called after removing a candidate from the election. |
| + RemoveCandidate(name: string) : Candidate | Removes the candidate with the given name from a party and returns that candidate |
| + RunElection(void) : int | Calls GetElectionType() and then the appropriate function of either ComputeIRElection() or ComputeOPLElection() based on the election type. |

## Report

| | |
|---|---|
| + WriteIRAuditReport(election) : int | Saves the Instant Runoff election results to an audit file, returning a 0 for success or 1 for failure |
| + WriteIRMediaReport(election) : int | Saves the Instant Runoff election results to a media report file, returning a 0 for success or 1 for failure |
| + WriteIRResultsToScreen(election) : int | Prints the results of an Instant Runoff election to the terminal, returning a 0 for success or 1 for failure |
| + WriteOPLAuditReport(election) : int | Saves the Open Party election results to an audit file, returning a 0 for success or 1 for failure |
| + WriteOPLMediaReport(election) : int | Saves the Instant Runoff election results to a media report file, returning a 0 for success or 1 for failure |
| + WriteOPLResultsToScreen(election) : int | Prints the results of an Open Party election to the terminal, returning a 0 for success or 1 for failure |
| + Report(IR) : void | Calls IRWriteMediaReport(), IRWriteAuditReport(), and IRWriteResultsToScreen() to return all election information |

| | |
|---|---|
| + Report(OPL) : void | Calls OPLWriteMediaReport(), OPLWriteAuditReport(), and OPLWriteResultsToScreen() to return all election information |

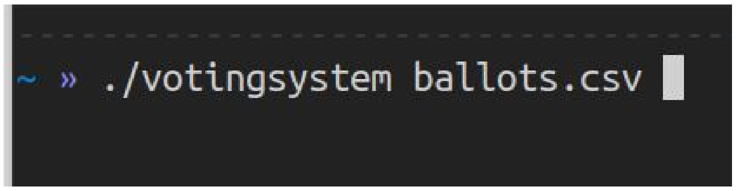
## 6. Human Interface Design

### 6.1 Overview of User Interface

The user interface will use the bash terminal to execute the Voting System. A lot of the user's interaction with the Voting System will be outside the scope of the program. They will create and modify the CSV input file using a tool of their choice, whether that's Excel, Google Sheets, or some command line tool. Various text editors will be able to read and modify the generated audit and report file. Then, users can execute the Voting System on this newly created CSV file to compute the appropriate (IR or OPL) election results.

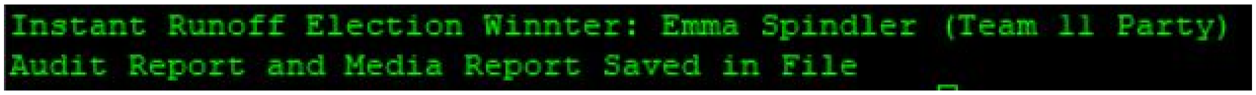
### 6.2 Screen Images

The user will be able to run the program using textual input seen below.

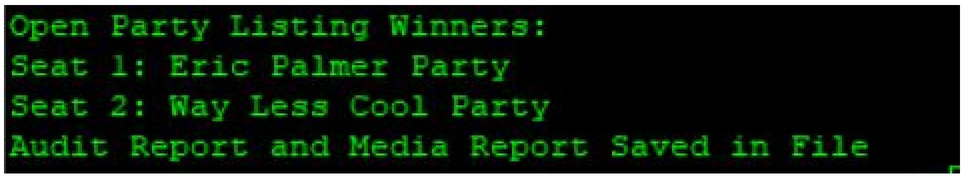


```
~ » ./votingsystem ballots.csv
```

The following results will be displayed once the Voting System Finishes Running dependent on the election type.



```
Instant Runoff Election Winnter: Emma Spindler (Team 11 Party)
Audit Report and Media Report Saved in File
```



```
Open Party Listing Winners:
Seat 1: Eric Palmer Party
Seat 2: Way Less Cool Party
Audit Report and Media Report Saved in File
```

### 6.3 Screen Objects and Actions

The only object on the screen will be the command line, which can be interacted in by typing the name of the program executable followed by a space and the path to a CSV file of appropriately formatted voting information.

## 7. Requirements Matrix

| Use Case | Relevant System Components |
|---|---|

| | |
|---|---|
| Coin Flip (CER_001) | + ResolveTie(num_candidates : int): int |
| Store Voting Information (ALBF_001) | Handled by election officials |
| Program Execution (CER_002) | Terminal Commands |
| Produce Audit Report (PER_001) | + OPLWriteAuditReport(void) : int<br>+ IRWriteAuditReport(void) : int |
| Compile Program (MISC_001) | Terminal Commands |
| Produce Media Report (PER_002) | + IRWriteMediaReport(void) : int<br>+ OPLWriteMediaReport(void) : int |
| Display Election Information in Terminal (PER_003) | + OPLWriteResultsToScreen(void) : int<br>+ IRWriteResultsToScreen(void) : int |
| Sharing Election Results with Media Officials (PER_004) | + IRWriteMediaReport(void) : int<br>+ OPLWriteMediaReport(void) : int |
| Load CSV File (ALBF_002) | + ReadInElectionType(void) : int<br>+ ReadInCandidates(void) : int<br>+ ReadInBallots(void) : int<br>+ ReadInNumberOfBallots(void) : int<br>+ ReadInNumberOfSeats(void) : int |
| Report Filename Convention (PER_005) | Convention for Naming the Files - These methods will be used when naming the file.<br>+ Report(OPL) : void<br>+ Report(IR) : void<br>+ OPLWriteMediaReport(void) : int<br>+ OPLWriteAuditReport(void) : int<br>+ IRWriteMediaReport(void) : int<br>+ IRWriteAuditReport(void) : int |
| Determining Election Type (CER_003) | + ReadInElectionType(void) : int |

## 8. Appendices

OPL_Activity_Diagram, IR_Activity_Diagram, Sequence Diagram, and the UML_Class_Diagram raw images are located in the SDD folder.