Team 19

Election Calculator Software

Software Design Document

Name (s):
Zach Ross,
rossx297@umn.edu
Zaid Nolley,
nolle044@umn.edu
Christopher Atherton
ather046@umn.edu

Workstation: KH1250-05

Date: (02/25/2024)

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

This software design document describes the architecture and system design of Election Calculator Software (ECS). It is intended to be read by election officials, clients, and the design and development team.

1.2 Scope

The ECS will allow election officials to automatically assign winners to open seats based on the results of an election. The ECS will do this by reading a specially formatted ballot, deciding what type of election it is, adding up the vote totals, then assigning winners to seats. The ECS will also decide ties via a fair coin flip, generate an audit file, and output the results to the election offical's graphical user interface (GUI).

1.3 Overview

This document outlines the ECS system. It does this with this written portion and three attached diagrams. A UML class diagram which shows all classes and their relationships. A sequence diagram for running the Open Party Listing (OPL) from start to finish. And finally a UML activity diagram (or process model).

1.4 Reference Material

https://standards.ieee.org/ieee/1016/1478/

Software Engineering 10th Edition, Ian Sommerville

1.5 Definitions and Acronyms

- Audit file - a file to track election results to be analyzed at a later date.
- Ballot - a form recording a voters candidate choices
- C++ - is a computer programming language
- Candidate - someone running for an elected office
- Command Line - is a method of communicating with a program
- CPL - or Closed Party Listing is a type of election system.
- CPU - or Computer Processing Unit is a hardware component of most computer systems.
- Data Structure - is a method to organize data in a program.
- Directory - is a part of a computer's file management system, usually containing other directories and/or files.
- Election - a method for selecting a candidate via voting.
- Executable Files - a file that can be run once prompted.
- Extract - to remove contents.
- GUI - or Graphical User Interface is a reason to interact with a program and for the program to display information.
- Linux Ubuntu 20.04 - is a type of operating system.

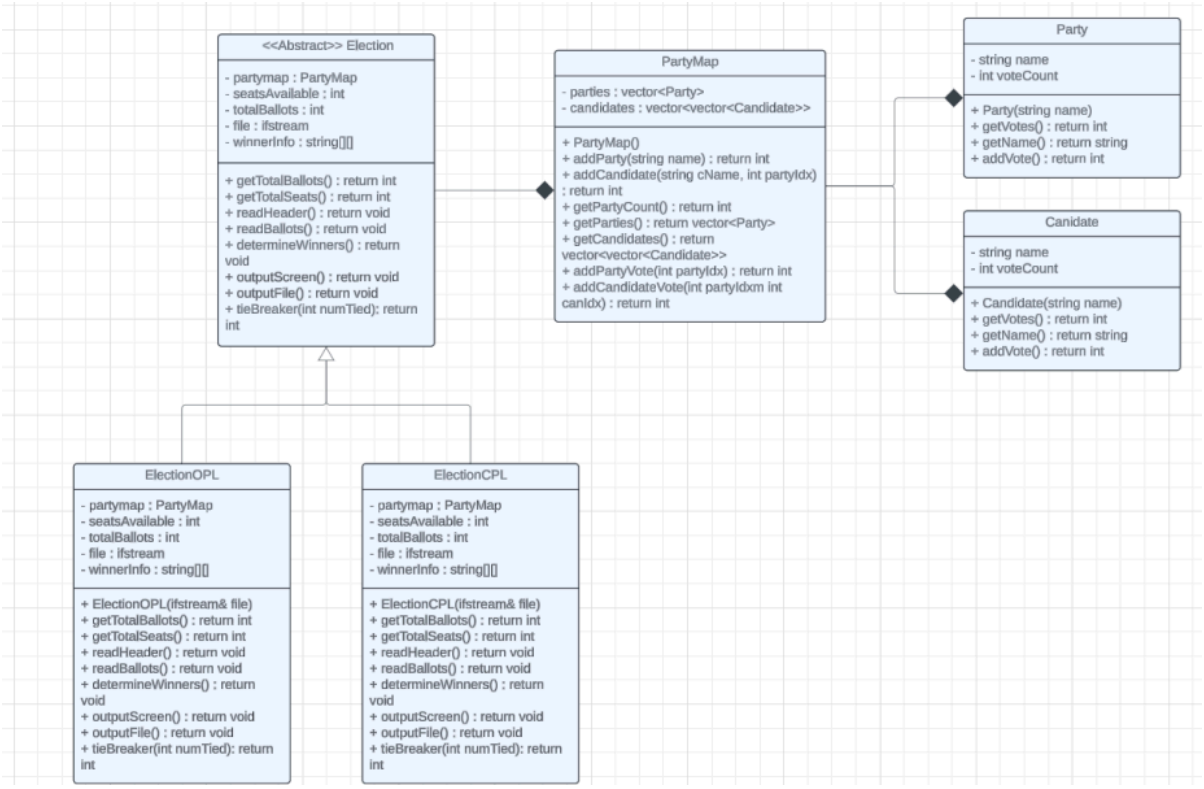
- Operating System - the software that handles most of the computer's functions, namely programs.
- OPL - or Open Party Listing is a type of election system.
- Path - relates to directory navigation. Directories link to one another this forms their “path”.
- Product Testers - often computer programmers who ensure a product in development works according to the agreed upon customer specifications.
- RAM - or Random Access Memory is a hardware component of most computer systems.
- README file - is a file describing useful or relevant information about a particular program, usually found in the program’s directory.
- Seat - a position that is filled with elected officials.
- Software - is a computer program.
- Terminal - is a program that utilizes command line prompts and inputs to allow users efficient access programs.
- UNIX - is a type of operating system.
- Use Case - is a specific scenario in which to apply a solution.

2. SYSTEM OVERVIEW

The ECS system is meant to allow the automatic calculation and assignment of seats based on the information provided by the ballot. The election official can input the ballot name via a GUI. If the ballot is in the same directory as the ECS and the filename is input correctly, the ECS will read the first line of the ballot to determine what type of election it is: Closed Party Listing (CPL) or Open Party Listing (OPL). The system will then calculate the votes by storing the totals in the specific election type’s data structure. Once the votes are tallied the system will then assign winners to open seats based on which election type is called for on the ballot’s first line. If a tie occurs the tie breaker system will automatically decide the winner using a fair random number generator. After all the seats have been assigned, the system will generate an audit file. Which will contain all the election results including non winning candidate’s results. All election results will also be displayed on the GUI.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

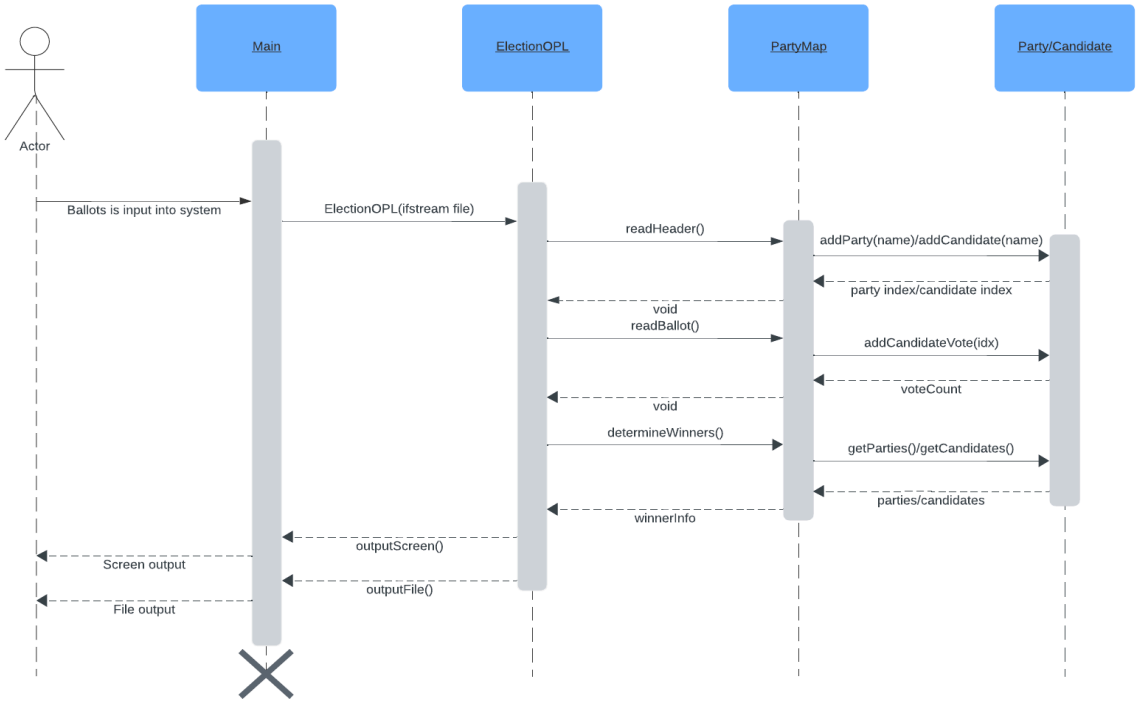


The ECS software is divided into a number of modules in order to simplify and effectively split responsibilities and allow for extensibility in line with the principles of object oriented design. The abstract *Election* class contains the functions that will be used in any election type; it's children are classes of specific election types (i.e. OPL and CPL) that either implement, override or extend the logic of those functions. These classes contain the bulk of the file I/O functionality for both the input ballot file and output audit file.

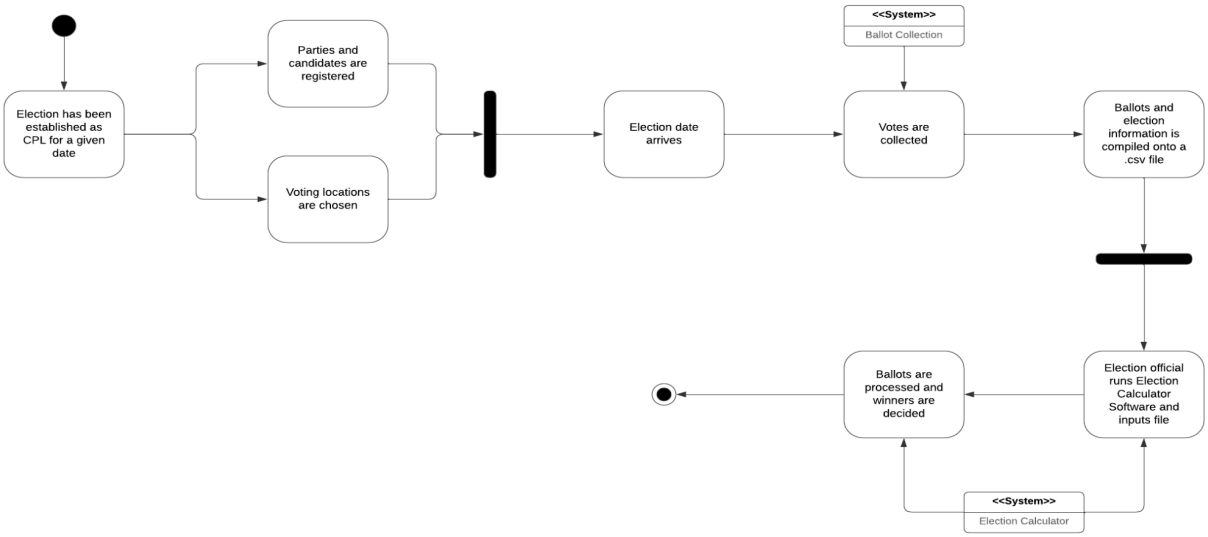
The *Election* class has a composition relationship with a module called *PartyMap* that in turn has a composition relationship with two more modules called *Party* and *Candidate*; these two classes primarily exist to hold the data that is being calculated in the *Election* classes and the *PartyMap* class is used to hold these classes and map the relationship between the candidates and their parties.

3.2 Decomposition Description

Sequence Diagram for OPL election processing:



Activity diagram of CPL election processing:



3.3 Design Rationale

The initial design architecture that was considered involved placing more of the election calculation logic inside of the *Party/Candidate* classes and also making these two classes have a more direct relationship with each other and also having some of the election calculation functionality, thinking that this would be a fairly intuitive design that would be easily implemented. The shift to the present design of having a *PartyMap* class was done because on closer inspection of the differences between OPL and CPL elections that it would not be feasible to put the election calculation functionality in any sort of super class or generic class.

The other notable feature of the software design is the implementation of the file input, initial ballot processing, and audit file creation inside of the *Election* class and its children rather than a separate main or file I/O class or classes. Creating separate modules to handle these features would likely be ideal if this software was to be extended to handle more types of elections, but in its present state, it is easier to represent these functionalities in the *Election* classes to more simply implement the different types of election and minimize the number of modules that contain election data.

4. DATA DESIGN

4.1 Data Description

The ECS system processes the information contained in ballot .CSV files that are passed to it The header of the ballot file is used to indicate the type of election is being calculated, the party information, and the candidate information. This information is passed to the *PartyMap* class which initializes the *Party* and *Candidate* classes that hold the information about the number of votes that they received individually. The *PartyMap* class maps this information and allows the *Election* classes to interpret the data it holds to ascertain election results that get passed into a 2d array of election results that is used to show election results to the users of software. The *Election* classes also pass the information that is stored in the *PartyMap* class to an output audit file that is stored in the local directory.

4.2 Data Dictionary

Election class methods, parameters, and return types:

- determineWinners(): void
- getTotalBallots(): int
- getTotalSeats(): int

- outputFile(): void
- outputScreen(): void
- readBallots(): void
- readHeader(): void
- tieBreaker(numTied: int): int

Election Class attributes:

- file: ifstream
- partymap: PartyMap
- seatsAvailable: int
- totalBallots: int
- winnerInfo: std::string[][]

ElectionOPL and ElectionCPL classes are identical to the Election class except each has their own constructor:

- ElectionOPL(file: ifstream&)
- ElectionCPL(file: ifstream&)

Party class methods, parameters, and return types:

- addVote(): int
- getName(): std::string
- getVotes(): int
- Party(name: std::string)

Party class attributes:

- name: std::string
- voteCount: int

Candidate class methods, parameters, and return types:

- addVote(): int
- Candidate(name: std::string)
- getName(): std::string
- getVotes(): int

Candidate class attributes:

- name: std::string
- voteCount: int

PartyMap class methods, parameters, and return types:

- addCandidate(cName: std::string, partyIdx: int): int
- addCandidateVote(partyIdxm: int, canindx: int): int
- addParty(name: std::string): int
- addPartyVote(partyIdx: int): int
- getCandidates(): vector <vector <Candidate>>
- getParties(): vector <Party>
- getPartyCount(): int
- PartyMap()

PartyMap attributes:

- candidates: vector <vector <Candidate>>
- parties: vector <Party>

5. COMPONENT DESIGN

```
// On program launch
FILE* file;
toScreen (“Input filename here: (string fileName = getInput())”);
file = openFile (fileName); //will create a pointer to filename
While (file != NULL)
        toScreen (“Check filename spelling. Input filename here: (fileName = getInput”));
        file = openFile (fileName); //will create a pointer to filename
        string electionType = Election::readHeader();
if (electionType = “OPL”) // all that follows is called from ElectionOPL
        getTotalSeats(file); // creates a list of seats available
        getBallots(file); // creates a list candidates
                //addCandidate() used to add candidates to the list of candidates in OPL
        readBallots(file); //counts votes adding them to the candidate
                //addCandidatesVote(); //adds votes to candidates list
        determineWinner(); // creates a list of winners and determines winners one at a time
        updating the available seats list, calls tieBreaker() if a tie is encountered.
```

```
        outputScreen(); // displays the results to screen
        outputFile(); //creates the audit file in the current directory.
else (electionType = “CPL”) //all that follows is called from ElectionCPL
        getTotalSeats(file); // creates a list of seats available
                //addParty() used to add parties to the list of parties in CPL
        getBallots(file); // creates a list parties
        readBallots(file); //counts votes adding them to the candidate
                //addPartyVote(); //adds votes to party list
        determineWinner(); // creates a list of winners and determines winners one at a time
        //updating the available seats list, calls tieBreaker() if a tie is encountered.
        outputScreen(); // displays the results to screen
        outputFile(); //creates the audit file in the current directory.
```

PartyMap // used to create the data structures used

6. HUMAN INTERFACE DESIGN

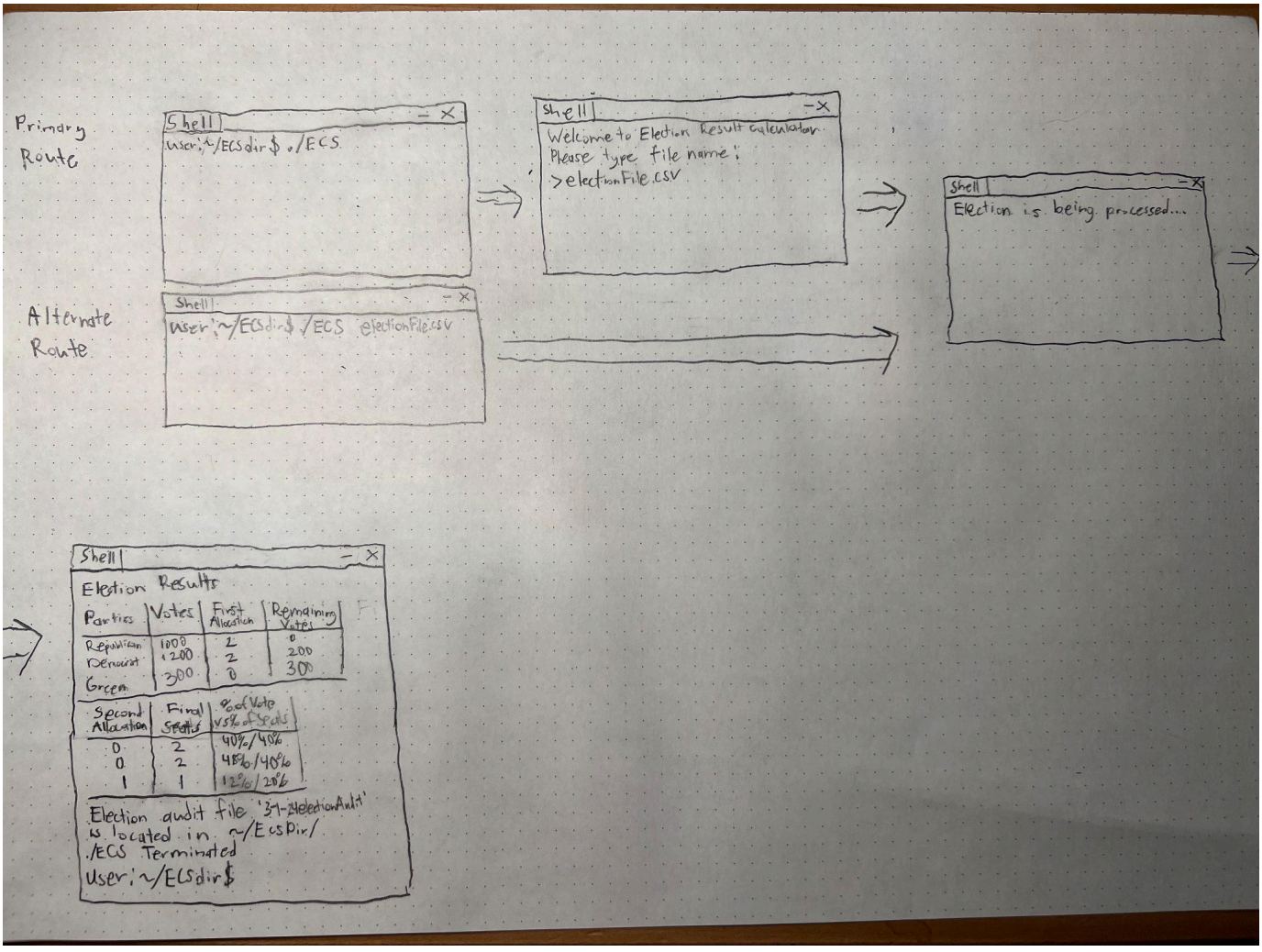
6.1 Overview of User Interface

The user has two possible courses of action in order to use the ECS software that ultimately join together to produce the same result. The first being the option to pass the ballot .CSV file in as an argument while launching the software and the second is launching the software without any additional arguments. In the case of the former the user will be met with a message on the terminal indicating that the election results are being calculated. In the case of the latter, the user is met with a message on the terminal prompting them to type in the file name of the ballot .CSV file and press enter or return; after this the user is met with the same message as the former user and both cases proceed identically.

Once the election results have been fully calculated, the terminal displays a graphical representation of the results in the form of a table with headers indicating what each column contains, i.e. parties, votes, seats allocated in the first round, remaining votes, followed by seat allocations and remaining votes for any number of subsequent rounds, and finally the final seat allocations and total percentage of votes received compared to the percentage of seats won.

Below the table of results the user will be shown a message indicating that an audit file has been created along with its name and location in the file directory. Finally a message is displayed indicating that the program has exited and the user is returned to the system terminal that they had been on initially.

6.2 Screen Images



6.3 Screen Objects and Actions

The first screen object displayed on launch of the program will be “Input filename here: ” This is produced by the Elections class main. Inputting the correct file name will launch the getHeader(). Otherwise the prompt will read “Unknown filename. Check the spelling of the

filename and enter it here: ”. This will permanently loop until the correct filename is input. This is also handled by Election class main.

If the header is OPL then the system will display the results of the election according to OPL. This will be handled by the ElectionOPL class.

If the header is CPL then the system will display the results of the election according to CPL. This will be handled by the ElectionCPL class.

7. REQUIREMENTS MATRIX

UC_001	Election getHeader, Election readBallot
UC_002	Election file : ifstream
UC_003	Election file : ifstream
UC_004	ElectionCPL tieBreaker
UC_005	ElectionOPL tieBreaker
UC_006	Election readHeader
UC_007	Election determineWinner
UC_008	ElectionOPL outputFile, ElectionCPL outputFile
UC_009	ElectionOPL outputScreen, ElectionCPL outputScreen
UC_010	Election readHeader
UC_011	ElectionOPL determineWinners
UC_012	ElectionCPL determineWinners

8. APPENDICES

Attachment_1 : is a UML class diagram for the entire system.

Attachment_2 : is a Sequence diagram for running the OPL election, starting from the moment you start processing the file and ballots to the end of declaring the winner(s).

Attachment_3 : UML activity diagram for the CPL election from start to finish.

Attachment_4: Hand drawn image of the user GUI interface.