Voting System

Software Design Document - **Team 24**

Name (s): Dahir Ali, Matt Graba, Julian Heyman, Muhsin Mohamed

Date: (03/03/2023)

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

This Software Design Document describes the architectural and system design of the voting system. The intended audience are the developers who need to design and build the system and documentation writers to create the Software Design Document.

1.2 Scope

This document contains a complete overview of the ballot processing system for running elections. Election officials or testers will be able to run the program which will automatically process a provided ballot file to determine the winner for CPL and IR elections

1.3 Overview

This document is intended to explain the system design and overview of the steps that need to occur to go from an election occurring to ultimately a winner being decided, or in the case of a CPL multiple “winners”. This document contains 5 main sections, the system architecture section, which explains the processes needed for this system to function. The data design section to explain the data structures used. The component design which discusses the individual components of our system. The human interface design which discusses what parts of the system a human user can interact with and how that will lok. And finally the requirements matrix which traces the SRS requirements to our components and data structures

1.4 Reference Material

- Software Specification requirements Document (SRS)
- Use Case Document

- Software Design Document Example
- Instant Runoff(IR) Activity Diagram
- Closed Party Listing(CPL) sequence Diagram
- Class Diagram of Entire System

1.5 Definitions and Acronyms

- CPL/IR: Closed Party Listing and Instant Runoff are the 2 forms of voting this system accepts. In a CPL people are allowed to vote for 1 party and then for however many votes a party receives they receive a proportional amount of votes with any remaining seats going to the party with most remaining votes. In an instant runoff people rank the candidates and then candidates with the least number of votes are eliminated until a candidate has a majority
- SDD: Software Design Document
- SRS: Software Requirements Specification
- Audit file: File containing election information on who won and how the election progressed

2. System Overview

Our project is to design a system that can read in a provided ballot and then process it according to the CPL or IR algorithm to produce a winner or winners. As of the first iteration the provided ballot file is assumed to not have any errors. The file can either be passed in as the program is started or the user will be prompted for the file after start up. Furthermore the program can only record a single election at a time, so it can't handle more than one and an entire election needs to be on the single ballot.

3. System Architecture

3.1 Architectural Design

Five main subsystems are composed to divide the modular program structure. These subsystems include the input/output, the ballot processing, the vote calculation, the audit logging, and the results reporting. The communication between the user and the software is handled by the input/output subsystem. It accepts files containing voter data and election parameters as input, and it outputs the election results to a file. The ballot file that was received from the Input/Output subsystem must be processed by the Ballot Processing subsystem. The ballots are read in, sorted by constituency, and then sent to the subsystem that calculates votes. Based on the sorted ballots that the Ballot Processing subsystem sent, the Vote Calculation subsystem is in charge of calculating the votes for each party in each constituency. The Audit Logging and Results Reporting subsystems receive the computed results after that. All information that is provided by the ballot file must be recorded by the Audit Logging

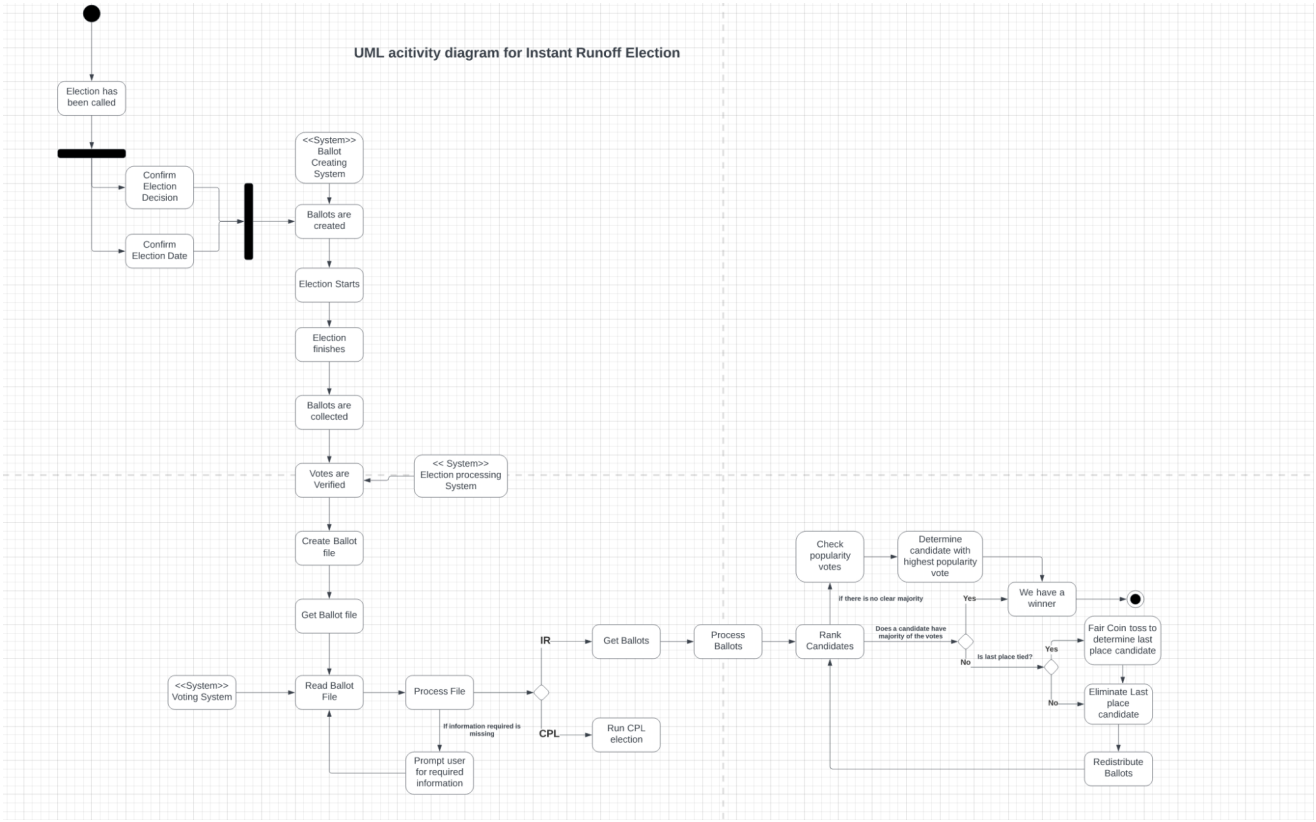
subsystem. This data is saved in an audit file. The Vote Calculation subsystem's results are compiled by the Results Reporting subsystem, which also creates a report of the election results that is output to a file.

There are multiple factors that contribute to the interconnection between these subsystems. For starters, receiving the input ballot file and sending the output results file requires communication between the input/output subsystem and the ballot processing subsystem. To convey the sorted ballots, the Ballot Processing subsystem interfaces with the Vote Calculation subsystem. In order to report the calculated results to the Audit Logging subsystem and to receive the official election results, the Vote Calculation subsystem connects with both of these systems. All election-related information is logged by the Audit Logging subsystem, which takes input from the Vote Calculation subsystem. The Vote Calculation subsystem provides information to the Results Reporting subsystem, which then produces the final report of the election results.

In theory, the modular program structure is created to effectively divide the system's duties into various subsystems to ensure a distinct separation of concerns. The Input/Output subsystem manages user interaction, the Ballot Processing subsystem sorts the ballots, the Vote Calculation subsystem computes the votes, the Audit Logging subsystem record all election-related details, and the Results Reporting subsystem creates the official report of the election results. To ensure the system functions as it should, these subsystems work together by sharing data.

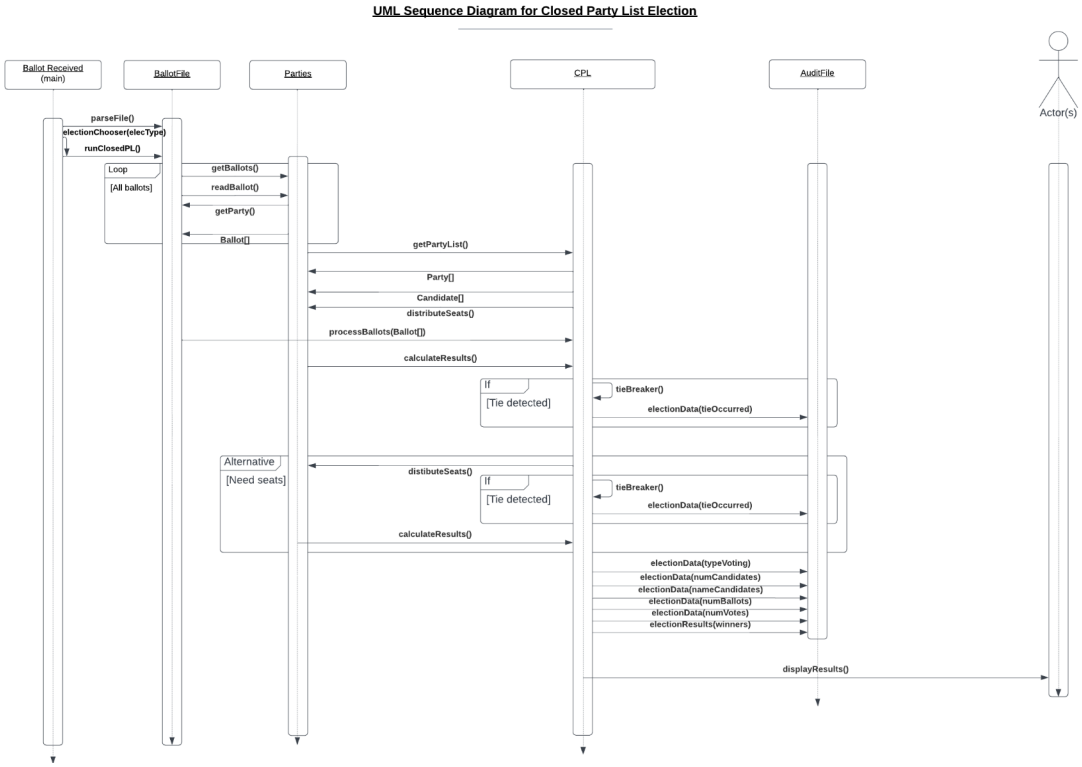
3.2 Decomposition Description

Instant Runoff Election Activity Diagram:



Once an Election has been called, The election date and decision needs to be confirmed. The ballots are created using the Ballot creating system and then election starts. Once the election is complete, the ballots are collected and verified using the election processing system. The Ballot file is created and retrieved with the information about the election. The file is then read in and processed using the voting system being implemented.If required information is missing then we prompt the user for required ballot information and continue to read and process the file. If the election is a Closed party listing Election, it runs the CPL election process. If the election is an instant run off election, the ballots are gathered and processed from the file. The candidates are then ranked. If a candidate has a majority of the votes, we determine a winner and the process ends. If there is no majority, check to see if last place tied. If tied, implement a fair coin toss to determine who is eliminated. If it is not tied, eliminate last place. Once last place is eliminated, redistribute their ballots to their next choice and continue to run that process. If there is no clear majority, check the popularity vote and determine the candidate with the highest popularity vote. That candidate is then determined the winner

Closed Party List (CPL) Election Sequence Diagram:



Beginning with the instant the system receives the ballots from the file, the sequence diagram for CPL is broken down as follow:

In order to extract the list of candidates and the corresponding parties for each constituency, the system scans the ballot file. This data is supplied to the CPL Election object via the Ballot File object. The Ballot File object provides the list of parties to the CPL Election object. A list of the candidates for the election is sent by the Parties object to the CPL Election object. The list of parties and the related candidate lists are used to initialize the Parties object in the CPL Election object. Each ballot is processed by the CPL Election object, which identifies the party and candidate selections before adding the vote to the appropriate Party object. The CPL Election object determines the overall number of votes for each Party object after all ballots have been processed. If a tie occurs, the CPL Election object searches for ties and uses a tie-breaking algorithm to break them. The winners are then added to the list of winners. Finally results are then written to the audit file.

In summary, the CPL sequence diagram is divided into four objects: the Ballot File object, the Parties object, the CPL election object, and the Audit File object. The essential election information, such as the list of candidates, parties, and the number of seats for each constituency, is provided by the ballot file object. The CPL Election object receives a list of parties from the Parties object. For each criteria

such as processing the ballots, resolving any ties, and writing the results to the Audit File object, the CPL Election object conducts the election process. Which brings us finally to the Audit File object which will keep the election results for later use.

3.3 Design Rationale

Many factors led to the selection of the modular program structure. First of all, it enables the system to be divided into more manageable, smaller subsystems that can be independently built and tested. By lowering the possibility of errors and making the system simpler to manage, this can raise the system's overall quality. Also, modular architecture can make it easier to reuse code. This implies that a certain component or subsystem can be readily withdrawn and utilized again if it is required in another system. Long-term, this can save both time and resources. The system's scalability had a significant role in the decision to employ a modular architecture. It is simpler to introduce new features or functionality by segmenting the system into smaller pieces without having to make significant changes to the existing code.

The necessity for security, reliability, and performance are a few of the crucial aspects that were taken into consideration. Because of the modular architecture, security may be applied at the subsystem level, reducing the likelihood of vulnerabilities. Also, it is simpler to find and address potential problems by disassembling the system into smaller pieces.

In result, the modular program structure was chosen because it allows for simple scaling, encourages code reuse, and makes it easier to build and test subsystems independently. Important considerations including security and dependability.

4. DATA DESIGN

4.1 Data Description

This system does not use any form of database or saving of data, outside of what gets outputted to the audit file. Instead our system has 1 interface and then 2 classes that extend from it, one for IR one for CPL. within each one a temporary list will be created with each index representing a ballots results. Various other information such as the voting type, candidates/parties, number of candidates will be stored as local variables to the system

4.2 Data Dictionary

- Audit File Object - implements the methods for transferring election information
- Ballot File Object -
- Closed Party Listing Object - implements CPL election processes
- calculateResults() - calculates the results of the election
- displayResults() - displays the results of the election to the user
- electionChooser(elecType) - Chooses which election to run based on parameter
    - Parameters: elecType - type of election to be run (CPL is 1, IR is 2)
- electionData(elecType) - returns election type
    - Parameters: elecType - type of election to be run (CPL is 1, IR is 2)
- electionData(numCandidates) - returns number of candidates
    - Parameters: numCandidates
- electionData(nameCandidates) - returns names of candidates
    - Parameter: nameCandidates
- electionData(numBallots) - returns number of ballots
    - Parameter: numBallots
- electionData(numVotes) - returns number of votes
    - Parameter: numVotes
- electionResults(winners) - returns results of election:
    - parameter: winners
- electionData(noMajority) - Notifies the system if there is no majority
    - Parameter: noMajority - checks if there is no majority
- electionData(tieOccured) - Notifies the system if a tie occurs
- getBallots() - gets the ballots from the file
- getParty() - gets the party/parties from the file
- handlePopularity() - determines candidate with highest popularity vote
- notEnoughinfo() - checks to see if the file has enough information
- parseFile() - Finds and identifies the file and reads it
- Parties Object
- processBallots(Ballot[]) - processes the ballots from the ballot file
    - Ballot[] - array that holds the ballots
- promptUser() - asks user for information needed to continue the process
- readBallot() - reads the ballots
- runClosedPL() - runs the Closed Party list voting system
- runInstantRunoff() - runs the Instant Runoff voting system
- redistributeBallots() - function gives ballots to next choice in IR
- tieBreaker() - returns a new winner based on the tiebreaker method

5. COMPONENT DESIGN

1. **processBallots(Ballot[])**: All of the ballots that have been turned in for the election are stored in the ballot array. The array can hold up to MAX_BALLOTS number of votes with each entry representing a single ballot. When an array is initially constructed, it is empty. The addBallot function inserts a new ballot into the array while ensuring that the number of ballots is not above its limit. The ballot located at the given index in the array is returned by the getBallot method. The number of ballots currently in the array is returned by the getNumBallots function. The array's ballots are all removed using the clearBallots method.
2. **calculateResults():** The ballots in the ballot array are processed by the calculateResults() function, which also counts the number of votes for each party and flags any invalid ballots. The function determines the total number of seats to be distributed after all valid ballots have been counted, and then distributes seats to the parties in accordance with their vote totals. In order to display the results in descending order of seats, the function finally sorts the parties according to the number of seats allotted.
3. **displayResults():** Function displays the election winner and information about the election to the screen. The information includes the type of election, ballots, names and number of candidates.
4. **electionChooser(elecType):** takes in parameter elecType which is an integer. If the elecType is 1 the election type is CPL and if it is 2 then it is IR. If it is another number we prompt the user to pick either 1 or 2
5. **electionData(elecType),electionData(numCandidates),electionData(nameCandidates),electionData(numBallots),electionData(numVotes),electionResults(winners),electionData(noMajority),electionData(tieOccured):** These functions return information about the election if needed such as the audit file. It returns the election type, name and number of candidates, number of ballots and votes, winner of the elections. It also checks if there is no majority in an election in which a popular vote is used. The last functions checks if a tie has occurred
6. **getBallots():** The list of ballots is taken from the ballot file, parsed, and returned as an array of ballot objects by the getBallots() function. For each ballot that was read from the file, a new ballot object was then created and stored in an array. The array of ballot objects is then returned for additional processing.
7. **getParty():** The party object connected to a particular party name is retrieved using the getParty() function. The party object that matches to the input name is returned after providing a party name as input. It returns null if the input name cannot be found in the list of parties.
8. **handlePopularity():** Function determines candidate with highest popular vote and determines them the winner.
9. **notEnoughInfo() :** When there is not enough information to create and complete the voting

system such as votes/ballots and what type of election it is, notEnoughInfo() is run. It notifies the user that there is not enough information then promptUser() is run.

10. **parseFile():** The parseFile() method takes as input the path to a file, processes it, and extracts the number of constituencies, parties, and candidate lists for each party, together with the total number of votes cast in each constituency. It generates and initializes the required objects, including the Party and BallotFile instances, and inserts the pertinent information into them.
11. **promptUser():** The user is prompted for extra information needed for the voting process, such as the number of parties, their names, and the number of seats, using the promptUser() function. Before returning the data to the called function, it first verifies and stores the inputted data.
12. **readBallot():** The voter's selections for each race and candidate are read in one at a time by the readBallot() method. Each race and candidate are given the opportunity for the user to make a pick, which is subsequently saved in the corresponding element of the ballot array. After each choice has been made, the ballot array is returned.
13. **runClosedPL():** This function creates a voting system and runs it based on the Closed Party List method of determining an election. The system is prompted initially to pick an election type and when given a string that corresponds to this function it'll run.
14. **runInstantRunoff()** This function creates a voting system and runs it based on the Instant Runoff method of determining an election. The system is prompted initially to pick an election type and when given a string that corresponds to this function it'll run.
15. **redistributeBallots():** the function assigns votes from the last place candidate that was eliminated to their next choice
16. **tieBreaker():** in the case of a tie either between 2 candidates/parties or more for each tied person the code will start by selecting a random number for them, and then select a random number not associated with a party/candidate, whoever is closest to the random number wins the tie. In the case 2+ more parties/candidates are tied a new random number will be selected for them as well as the "goal" one

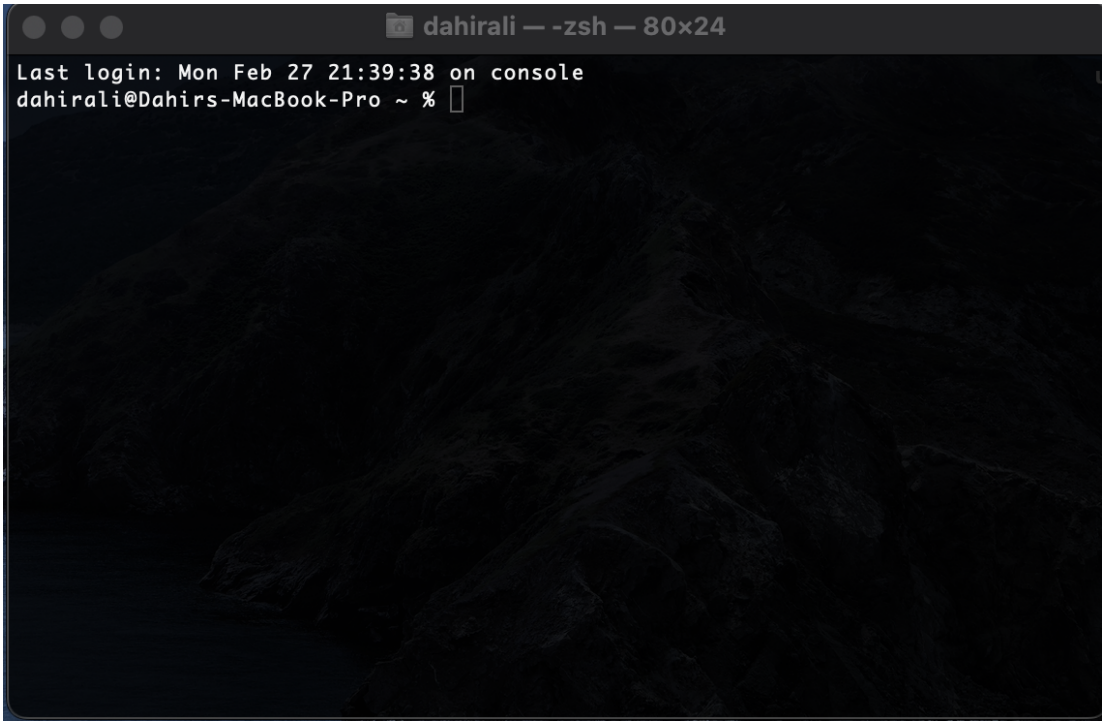
6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

The user has 2 different ways to start the system. The first is they can provide a file as an argument when starting the program, if that's the case the user will not be prompted for a file name unless the one they provided was invalid. The other course is they can start the program and then the system will print a prompt to the command window for a file name to be imputed.

Regardless of the path after the file has been accepted and opened the user will then be prompted to provide any needed information for the program to run if the file does not contain it, such as the voting type, number of ballots etc. after that the user won't get a new screen until a winner has been determined at which point they will both have information printed to the screen as well as an audit file produced

6.2 Screen Images



6.3 Screen Objects and Actions

The screen is the command line interface and the screen objects would be commands that the user can enter. Some of the actions associated include getting user input, running a file and returning output to the user

7. REQUIREMENTS MATRIX

SRS ID	Component

UC_001_identify_f	parseFile()
UC_002_read_f	getBallots(), readBallots(), getParty(), Ballot[], Party[]
UC_003_process_f	getPartyList(), processBallots(Ballot[]), calculateResults()
UC_004_handle_ir	electionChooser(elecType), runInstantRunoff()
UC_005_handle_cpl	electionChooser(elecType), runClosedPL()
UC_006_display_r	displayResults()
UC_007_produce_audit_file	electionData(elecType), electionData(numCandidates), electionData(nameCandidates), electionData(numBallots), electionData(numVotes), electionResults(winners)
UC_008_handle_no_ballot_number	notEnoughInfo(), promptUser()
UC_009_handle_popularity_v	electionData(noMajority), handlePopularity()
UC_010_tie_breaker	electionData(tieOccured), tieBreaker()
UC_011_ prompt_user_extra_info	notEnoughInfo(), promptUser()

8. APPENDICES

No Appendices at this moment.