Team 13

Project Waterfall

Software Design Document

Names: Amy Nguyen, Kaley Schiffler, Hoin Jang, William Henning
Lecture Section: 001

Date: (03/04/2022)

TABLE OF CONTENTS 1

1. INTRODUCTION

1.1 Purpose

This Software Design Document provides the architecture and system design of Project Waterfall. The expected audience is primarily election system researchers and developers of Project Waterfall with a secondary focus on the professor and teaching assistants of 5801W.

1.2 Scope

This document contains a complete description of the design of Project Waterfall. The software product will be developed using C/C++. The program is expected to run on a University of Minnesota CSE Lab machine using the command line.

Users will be able to run a fair Instant Run-off Voting or Open Party Listing election with a formatted CSV file and get audited results from the election, as well as a simplified text file of the results for media personnel. The benefit to this product will be the ability to conduct a fair election in a short amount of time.

1.3 Overview

The remaining sections of this document and their contents are listed below.

Section 2 is a description of the system overview. It contains the functionality, context, and design of the project.

Section 3 is the System Architecture that specifies the Architectural Design, Decomposition Description, and Design Rationale. The relationship between the modules and how they collaborate to achieve a fully functional system is described. Diagrams will be included to aid the descriptions.

Section 4 concerns the data design. This includes the data storage mechanics, system entities, and major data.

Section 5 provides pseudocode descriptions of all the system components.

Section 6 provides an image and description of the user interface.

Section 7 shows a tabularly formatted cross-reference of system components and the project SRS.

1.4 Reference Material

[IEEE] The applicable IEEE standards are published in “IEEE Standards Collection,” 2001 edition.

[Sommerville] Software Engineering by Ian Sommerville, 10th edition.

1.5 Definitions and Acronyms

IRV	Instant Run-off Voting
OPL	Open Party Listing
CSV	Comma-separated values, basically a formatted .txt file
IEEE	Institute of Electrical and Electronics Engineers

2. System Overview

Users will be able to indicate whether or not they wish to proceed with an IRV or OPL type election. This is indicated, along with some other information, at the top of the CSV file that is required as input and contains all the ballot and voting information. Once an election type has been determined, the system will continue with the selected election and the CSV file. The system will then count the votes accordingly until completion and output the results and other information relating to the election on the screen.

3. System Architecture

3.1 Architectural Design

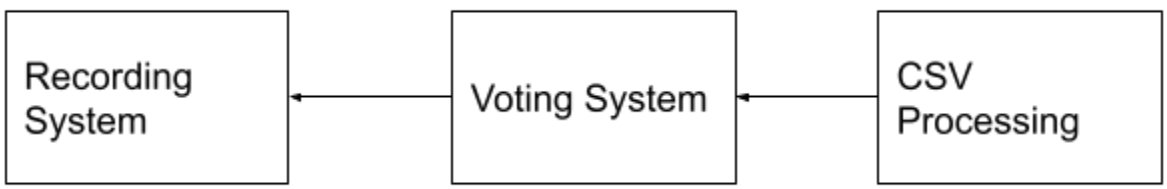


Figure 1: Subsystem Diagram

The CSV Processing subsystem handles the input of data, allowing for the user to input ballots in a CSV, processing them into a usable format, and passing that data into the voting system.

The Voting System finds the winner of the election based on the given ballots. This is split further into a system that handles OPL elections and a system that handles IRV elections.

The Recording System records the relevant information from the voting system and outputs it into an audit file to be verified by the users and displays the election results to the screen.

These subsystems and their activities can be shown in detail in Figures 5 and 6 of section 3.2. A more detailed explanation of the design rationale is provided in section 3.3.

3.2 Decomposition Description

(See diagrams below)

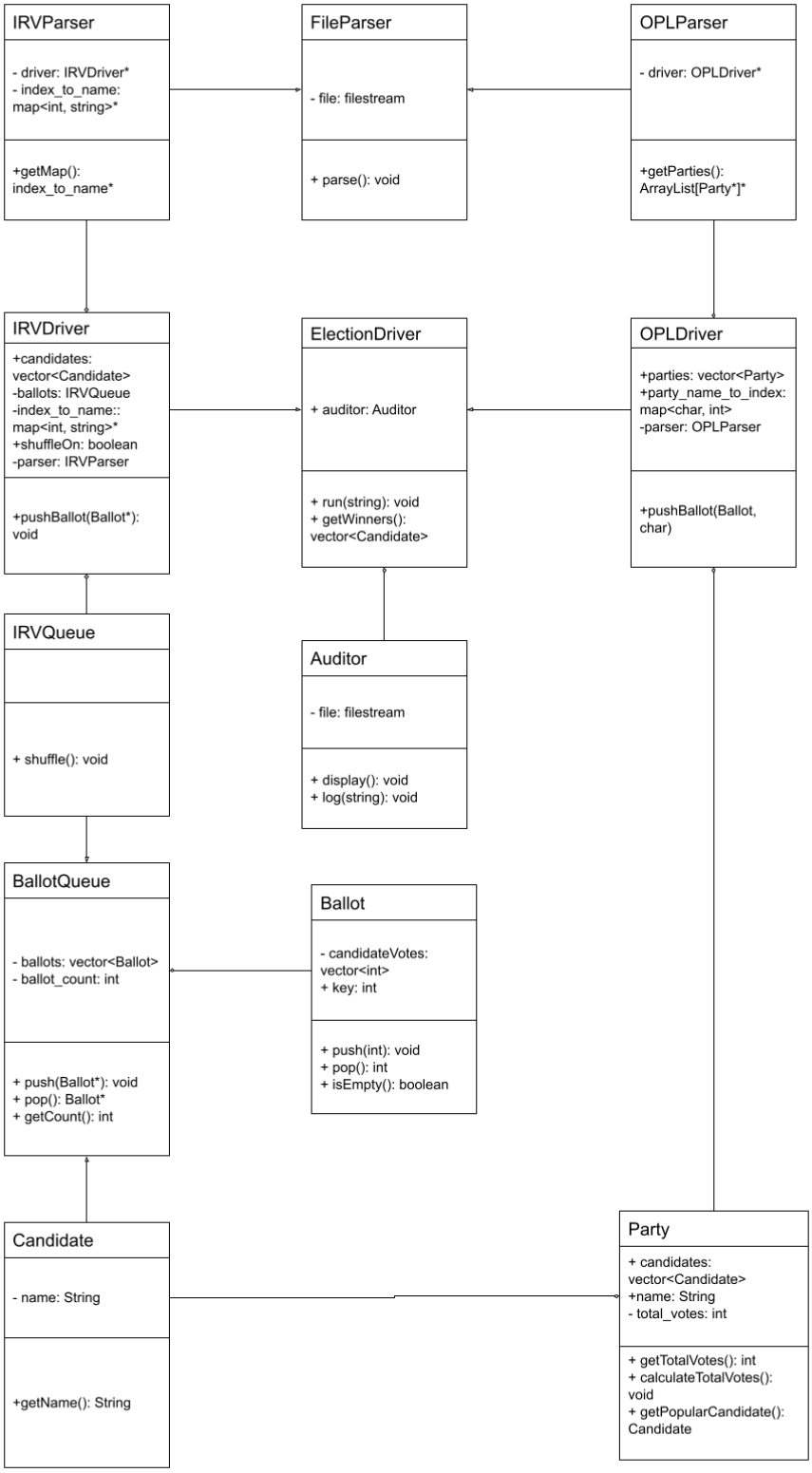


Figure 2. UML Class Diagram for the full system.

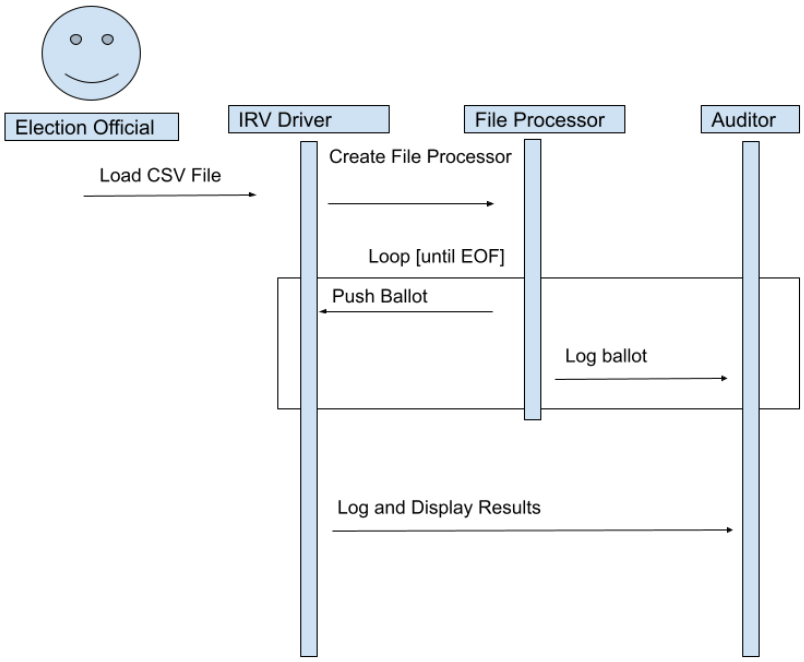


Figure 3: IRV Sequence Diagram

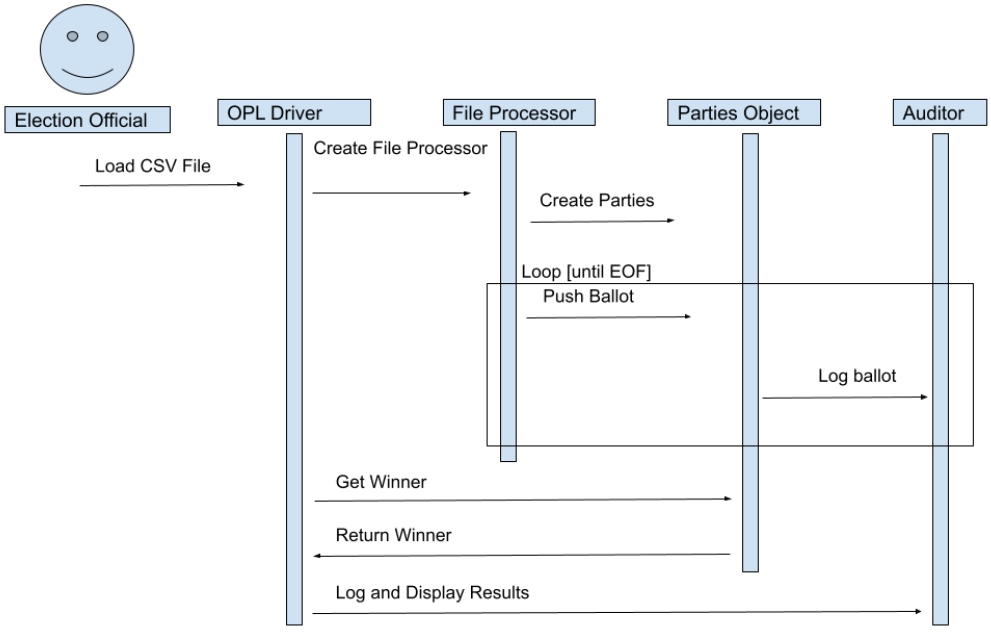


Figure 4: OPL Sequence Diagram

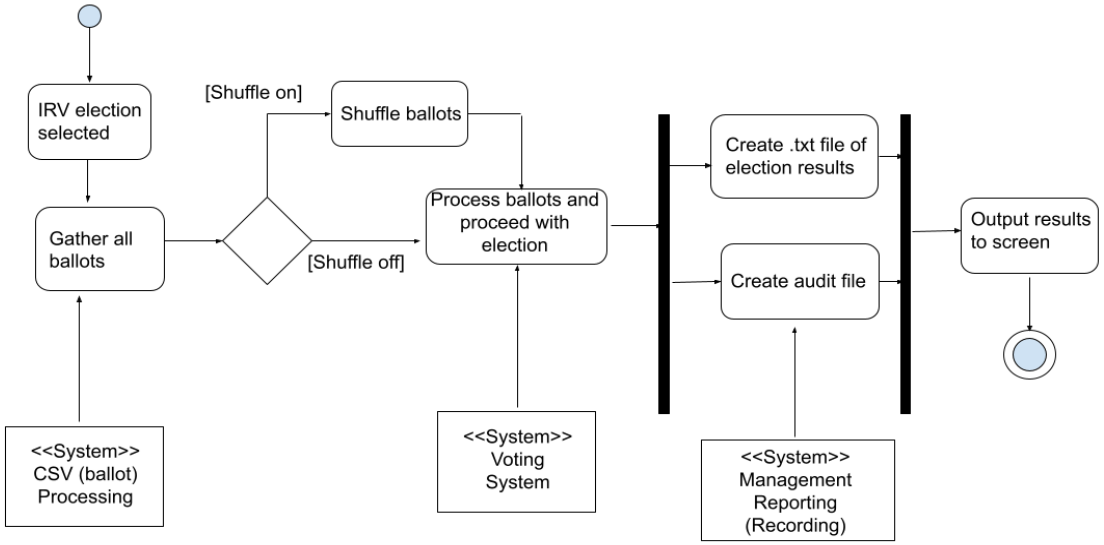


Figure 5: IRV Activity Diagram

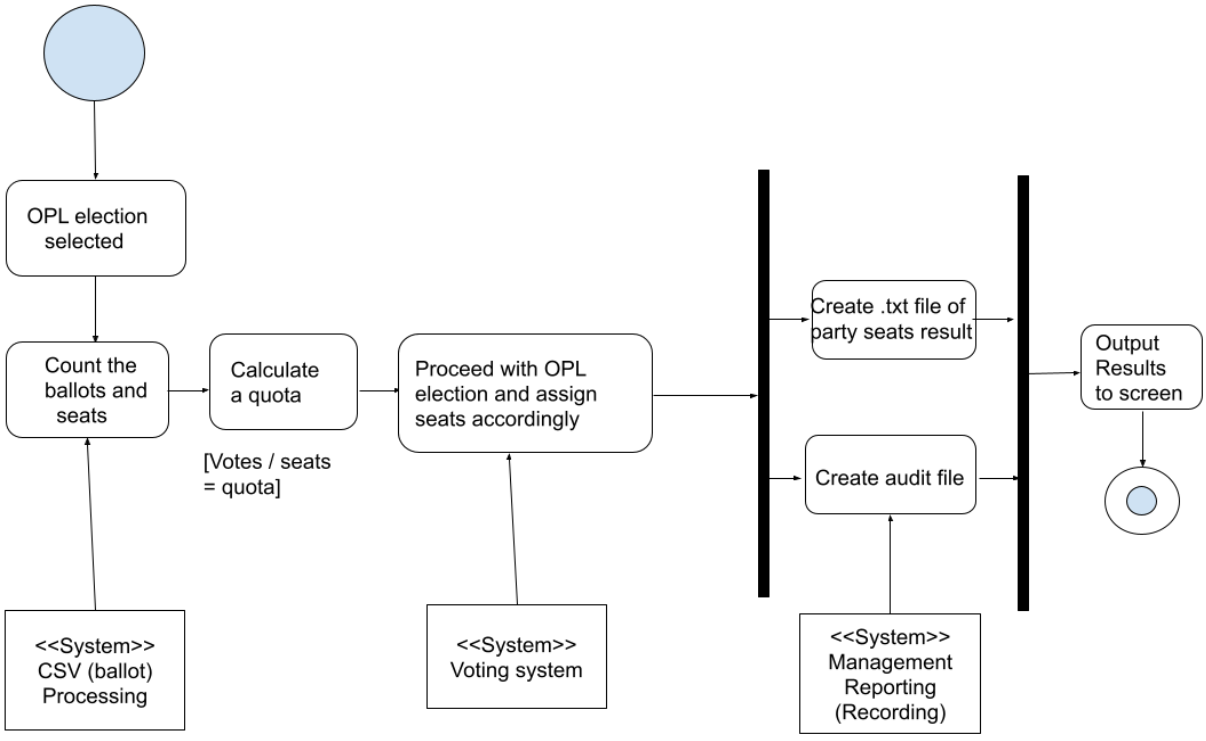


Figure 6: OPL Activity Diagram

3.3 Design Rationale

The first subsystem created was the CSV Processing System to gather all the ballots. This is needed to process the file that is used as input for the system and will determine if the election type is OPL or IRV. A potential issue in this subsystem would be a file not specifying which type of election the ballots are for. This would cause an error and prevents the rest of the overall system from running.

The second subsystem created was the Voting System. This processes the ballot information based on the election type and determines the winners of an election. This subsystem will do a majority of the work for the system and is divided into two parts, one for an OPL election and the other for an IRV election. We chose to have a subsystem that handles the entire voting process of an election in one place. We considered doing each election type as a different subsystem but realized this method could result in redundant code and increased complexity.

The last subsystem that we created was a Recording System that records the results and relevant data for the election and outputs that information to an audit file. Having a system that records the information is necessary in case the user or election official needs to verify the validity of the election. The user will have the means to recreate the election and prove the same result as the original election.

4. Data Design

4.1 Data Description

	Field	Type	Size
ElectionDriver	auditor	Auditor	32 (pointer)
IRVDriver	candidates	vector<Candidate>	32 (pointer)
	ballots	IRVQueue	32 (pointer)
	index_to_name	map<int, string>*	32 (pointer)
	shuffleOn	boolean	1
	parser	IRVParser	
OPLDriver	parties	vector<Party>	32 (pointer)
	party_name_to_index	map<char, int>	32 (pointer)
	parser	OPLParser	
FileParser	file	ifstream	

IRVParser	driver	IRVDriver*	32 (pointer)
	index_to_name	map<int, string>*	32 (pointer)
BallotQueue	ballots	vector<Ballots>	32 (pointer)
	ballot_count	int	4
OPLParser	driver	OPLDriver*	32 (pointer)
Ballot	candidateVotes	vector<int>	32 (pointer)
	key	int	4
Candidate	name	string	
Party	candidates	vector<Candidate>	32(pointer)
	name	string	
	total_votes	int	4

4.2 Data Dictionary

Auditor: void. outputs audit files to the screen.

- File: filestream
- display(): void
- log(string): void

Candidates: Vector of registered candidates.

- name: string()
- getName(): string

File Parser: void. processes and parses input. Includes ballot-related objects.

- file: filestream
- parse(): void

IRV Driver: User-defined, ElectionDriver. Initiates an IRV election.

- candidates: vector<Candidate>
- ballots: IRVQueue
- index_to_name:: map<char, int>
- shuffleOn: boolean
- parser: IRVParser

BallotQueue: User-defined, IRVQueue. Vector of ballots as a queue.

- ballots: vector<Ballot>
- ballot_count: int
- push(Ballot*): void

- pop(): Ballot*
- getCount(): int

Ballot:

- candidateVotes: vector<int>
- key: int
- push(string): void
- pop(): int
- isEmpty(): boolean

IRV Queue: void. Shuffles the ballots if shuffle is toggled on.

- shuffle(): void

OPL Driver: User-defined, ElectionDriver. Initiates an OPL election.

- parties: vector<Party>
- party_name_to_index: map<char, int>
- parser: OPLParser
- pushBallot(Ballot, char)

Parties Object: Vector of registered parties.

- candidates: vector<Candidate>
- name: String
- total_votes: int
- getTotalVotes(): int
- calculateTotalVotes(): void
- getPopularCandidate(): Candidate

5. Component Design

5.1 IRVParser

```
parse(){
        //parses each line, updating the map and pushing ballots to the IRV driver.
}

getMap(){
        //return a pointer to the map
}
```

5.2 OPLParser

```
parse(){
        //parses each line, starting with the header to create the party to index map, and
        instantiating all the candidates within their respective parties in the parties vector of the
        OPLDriver
}
```

5.3 IRVDriver

```
run(String filename) {
        //instantiate a fileparser and auditor
        //calculate winners
}

getWinners(){
        //run the IRV election itself, transferring ballots between candidates when they are
        eliminated, logging all intermediate steps to the auditor. Return a vector of winning
        candidates when done.
}

pushBallot(Ballot*){
        //push the Ballot onto the ballots queue.
}
```

5.4 OPLDriver

```
pushBallot(Ballot, char){
        // push ballot to corresponding party (represented by the char) and candidate within that
        party (which is encoded within the ballot itself)
}
```

5.5 Ballot

```
Push(int i){
        //Push a new vote onto the stack, with i being the index of the candidate as represented
        in the CSV
}


Pop(){
        //get the first vote off the stack
}

isEmpty(){
        //Return true if there are no more votes to process in the stack, for determining whether
        the IRV Driver can delete this ballot if their last candidate is eliminated.
}
```

5.6 BallotQueue

```
push(Ballot*){
        //push the ballot object onto the ballots vector
        //update ballot_count
}

pop(){
        //pop the ballot object off the ballots vector
        //update ballot count
}

getCount(){
        //return ballot_count
}
```

5.7 IRVQueue

```
shuffle(){
        // shuffle the ballots
}
```

5.8 Candidate

```
getName(){
        // return the name of the registered candidate
}
```

5.9 Party

```
getTotalVotes(){
        // return the total numbers of vote
}

calculateTotalVotes(){
        // calculate the total number of votes for the party
}
getPopularCandidate(){

        // Return the candidates with the most votes
}
```

5.10 Auditor

```
display(candidates[] winners){
        //display the result of winners
}

log(string message){
        // log message in audit file
}
```

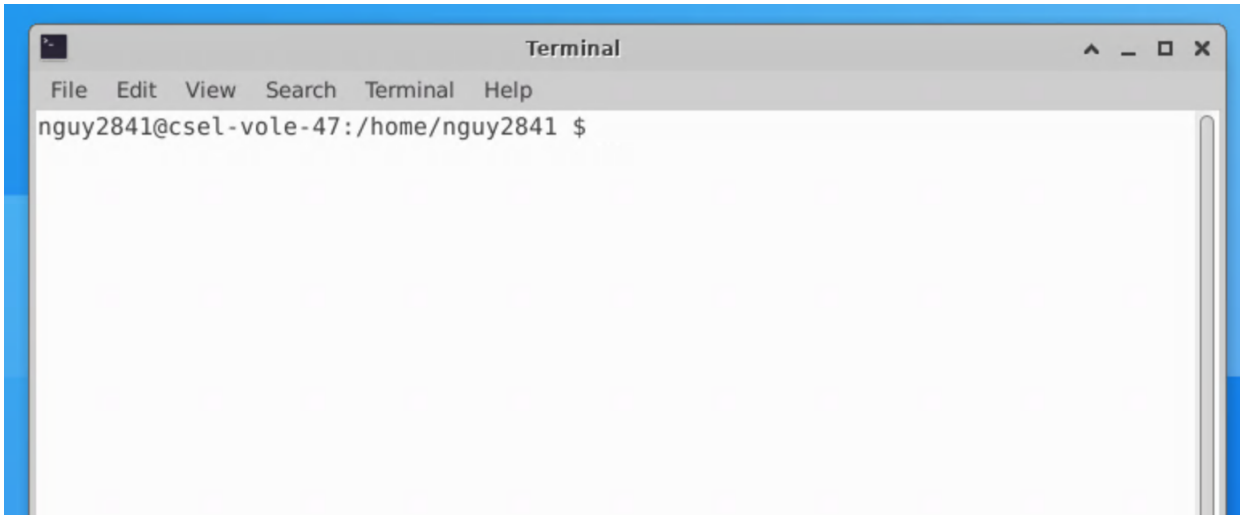
6. Human Interface Design

6.1 Overview of User Interface

The system will be used to determine the winner of an election. A user, most likely an election official, will be able to input ballots that have been filled in by the voters. The user will also input the party names and the candidate names. These ballots will be processed by the system and a .txt file will be generated for the user to view. An audit file will also be produced for the user to view. If the user is a tester, they have the option to turn off the shuffle function so that they can properly test the system.

6.2 Screen Images

This system will exclusively operate on a CSE lab machine terminal command prompt.



6.3 Screen Objects and Actions

The user types in the commands to run the system along with the input file and any other relevant information.

7. Requirements Matrix

System Components	SRS Functional Requirements
IRV Driver	UC_001 - Run IRV
OPL Driver	UC_002 - Run OPL
File Processor	UC_003 - CSV Input
Voting System	UC_004 - Toggle Ballot Shuffle UC_005 - View Election Results Section 4.7 - Coin flip
Auditor	UC_006 - Generate Audit File