Team 11

Voting System

Software Design Document

Name (s):
Chen Liu,
Jicheng Zhu,
Yingwen Weng,
Zilong He.

Date: (02/23/2021)

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Purpose

This software design document describes the architecture and system design of the voting system software.

The expected audience are election officials at voting stations, who want to use our voting systems for analyzing results of two types of voting (Instant Runoff Voting, IRV & Open Party List, OPL), Programmers and testers who are interested in working on the project by further developing it to be used for more types of voting or fix existing bugs.

1.2 Scope

This document contains a complete description of the design of our voting system.

The basic architecture is a voting system with the ability to handle two types of voting (instant runoff and open party list)

Team 11 members in charge of the voting system software can access to making changes when necessary. The change includes but is not limited to changing the codes of each section and fixing existing bugs.

1.3 Overview

This document is divided into seven sections. We go through the introduction of the document in section 1 and overview of the system in section 2.

In section 3, we describe the system architecture. Each entity contains a description concerning its functionality.

In section 4, we focus on the data design.

In section 5, components design of the system is discussed.

In section 6, we present the human interface design.

Finally, we present the requirement matrix in section 7 and appendix.

1.4 Reference Material

SRS document of team 11.

Software's Github:
https://github.umn.edu/umn-csci-5801-S21-002/repo-Team11.git

IEEE Template for System Requirement Specification Documents:
https://goo.gl/nsUFwy

Instant-runoff voting, Wikipedia:
https://en.wikipedia.org/wiki/Instant-runoff_voting?oldformat=true

Party-list proportional voting, Wikipedia:
https://en.wikipedia.org/wiki/Party-list_proportional_representation?oldformat=true

GUI, Wikipedia:
https://en.wikipedia.org/wiki/Graphical_user_interface?oldformat=true

RAM, Wikipedia:
https://en.wikipedia.org/wiki/Computer_memory?oldformat=true

CSV, Wikipedia:
https://en.wikipedia.org/wiki/Comma-separated_values?oldformat=true

API, Wikipedia:
https://en.wikipedia.org/wiki/API

1.5 Definitions and Acronyms

See Appendix A in SRS.

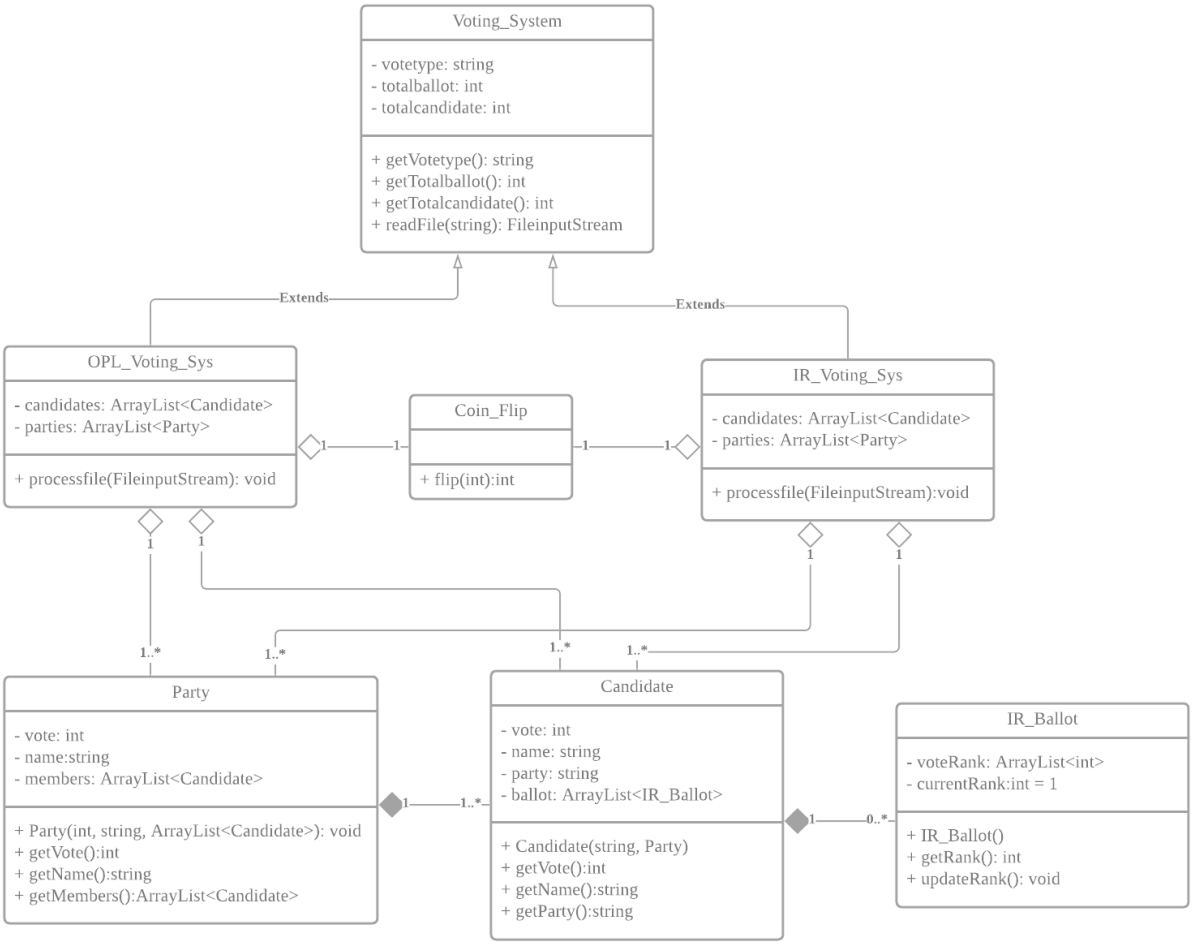
2. System Overview

See section I in SRS

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

Develop a modular program structure and explain the relationships between the modules to achieve the complete functionality of the system. This is a high level overview of how responsibilities of the system were partitioned and then assigned to subsystems. Identify each high level subsystem and the roles or responsibilities assigned to it. Describe how these subsystems collaborate with each other in order to achieve the desired functionality. Don't go into too much detail about the individual subsystems. The main purpose is to gain a general understanding of how and why the system was decomposed, and how the individual parts work together. Provide a diagram showing the major subsystems and data repositories and their interconnections. Describe the diagram if required.



3.2 Decomposition Description

Provide a decomposition of the subsystems in the architectural design. Supplement with text as needed. You may choose to give a functional description or an object oriented description. For a functional description, put top level data flow diagram (DFD) and structural decomposition diagrams. For an OO description, put subsystem model, object diagrams, generalization hierarchy diagram(s) (if any), aggregation hierarchy diagram(s) (if any), interface specifications, and sequence diagrams here.

Party
Name: Party
Type: Class
Description: This is for open party list voting (OPL), this class records all information about a candidate, including his or her name, party affiliation, number of voters, and ballot list.
Attributes: vote: int
name: string
members: ArrayList<Candidate>
Resources: None
Operations: New
Name: Party
Argument: int, string, ArrayList<Candidate>
Returns: Void
Pre-condition: candidate and party name is read as a string
Post-condition: create a new party object
Exceptions: None
Flow of Events:
1, pass initial int, string and Party into
2, return by creating a new party object

Operations: get value
Name: getVote()
Argument: None
Returns: int
Pre-condition: None
Post-condition: the number of voters of the candidate is returned
Exceptions: None
Flow of Events:
1, return the number of voters of the candidate

Name: getName()

Argument: None
Returns: string
Pre-condition: None
Post-condition: the name of candidates is returned
Exceptions: None
Flow of Events:
1, return the name of the candidate

Name: getParty()
Argument: None
Returns: string
Pre-condition: None
Post-condition: the party of the candidate is returned
Exceptions: None
Flow of Events:
1, return the party of the candidate

Candidate
Name: Candidate
Type: Class
Description: This is for open party list voting (OPL), this class records all information about a candidate, including his or her name, party affiliation, number of voters, and ballot list.
Attributes: vote: int
name: string
party: string
ballot: ArrayList<Ballot>
Resources: None
Operations:
Name: Candidate()
Argument: string, Party
Returns: None
Pre-condition: candidate and Party name is read as a string
Post-condition: create a new candidate
Exceptions: None
Flow of Events:
1, pass string and Party into
2, return by creating a new candidate

Operations: get value

Name: getVote()
Argument: None
Returns: int
Pre-condition:
Post-condition: the number of voters of the candidate is returned
Exceptions: None
Flow of Events:
1, return the number of voters of the candidate

Name: getName()
Argument: None
Returns: string
Pre-condition: None
Post-condition: the name of candidates is returned
Exceptions: None
Flow of Events:
1, return the name of the candidate

Name: getParty()
Argument: None
Returns: string
Pre-condition: None
Post-condition: the party of the candidate is returned
Exceptions: None
Flow of Events:
1, return the party of the candidate

Voting system

Name: Voting_System
Type: class
Descriptions: Parent class for OPL_Voting_Sys and IR_Voting_Sys. Store the general information including total ballot and total candidate. Determine the type of vote by reading the first line of the file. Ballot counting and other processing are done by the respective child class.
Attributes: votetype: string
totalballot: int
totalcandidate: int
Resources:None

Operations: get value

Name: getvotetype()

Arguments: None

Returns: string

Pre-condition: Successfully read the ballot file and the type of vote is stored in the attribute votetype.

Post-condition: Vote type is returned.

Exceptions: None

Flow of Events:

1, Return attribute votetype.

Name: getTotalballot()

Arguments: None

Returns: int

Pre-condition: Successfully read the ballot file and the number of ballots is stored in the attribute totalballot.

Post-condition: The number of ballots is returned

Exceptions: None

Flow of Events:

1, Return attribute totalballot.

Name: getTotalcandidate()

Arguments: None

Return: int

Pre-condition: Successfully read the ballot file and the number of candidates is stored in the attribute totalcandidate.

Post-condition: The number of candidates is returned.

Exceptions: None

Flow of Events:

1, Return attribute totalcandidate.

Name:readFile(string)

Arguments: string

Return: FileinputStream

Pre-condition: file name is read as a string

Post-condition: The content of the file is stored in a file stream. useful information is stored in respective attributes.

Exceptions: None

Flow of Events:

1, Use the string that contains filename to read the file and stored in a file stream.

2, The first line determines the type of vote, and stores it in the attribute votetype.

3, The second line has the number of candidates, store it in the attribute totalcandidate.

4, If it is IR, go to the fourth line and store the number in the attribute totalballot.
If it is OPL, go to the fifth line and store the number in the attribute totalballot.

OPL voting system

Name: OPL_Voting_Sys
Type: class
Description: The child class of Voting_System. Main class for performing OPL counting.
Attributes: candidates: ArrayList<Candidate>
parities: ArrayList<Party>
Operations: file processing
Name: processfile(FileinputStream)
Arguments: FileinputStream
Returns: None
Pre-condition: The type of vote is OPL
Post-condition: File finish processing and the winner(s) is declared.
Exceptions: None
Flow of Events:
1,

IR_Ballot

Name: IR_Ballot
Type: class
Descriptions: A specific class to record the data of the ballot. For each ballot, it contains a different rank of vote.
Attributes: voteRank:ArrayList<int>
currentRank:int
Resources:None
Operations: get value
Name: IR_Ballot()
Arguments: None
Returns: None
Pre-condition: None
Post-condition: current rank is set to one.
Exceptions: None
Flow of Events:
1, Return IR_Ballot object.

Name: getRank()

Arguments: None
Returns: int
Pre-condition: Successfully read the ballot file and the vote rank for each ballot is stored in the object class.
Post-condition: None
Exceptions: None
Flow of Events:
1, Return attribute currentRank.

Name: updateRank()
Arguments: None
Returns: void
Pre-condition: IR_Ballot object is initialized.
Post-condition: currentRank = currentRank + 1
Exceptions: None
Flow of Events:
1, Increase currentRank by 1
2, Return None.

IR_Volting_Sys
Name: IR_Voting_Sys
Type: class
Descriptions: The child class of Voting_System. Main class for performing IR runoff ballot.
Attributes: candidates: ArrayList<Candidate>
parties: ArrayList<Party>
Resources: None
Operations: file processing
Name: processfile()
Arguments: FileinputStream
Returns: None
Pre-condition: The type of vote is IR Ballot.
Post-condition: File finish processing and the winner(s) is declared.
Exceptions: None
Flow of Events:
1,

3.3 Design Rationale

Discuss the rationale for selecting the architecture described in 3.1 including critical issues and trade/offs that were considered. You may discuss other architectures that were considered, provided that you explain why you didn't choose them.

4. DATA DESIGN

4.1 Data Description

Party

Attribute name	Attribute type
vote	Int
name	string
Members	ArrayList<Candidate>

Vote: the number of votes a party has
Name: the name of the party
Members: a list of candidates belongs to this party

Candidate

Attribute name	Attribute type
vote	Int
name	string
party	string
ballot	ArrayList<IR_Ballot>

Vote: the number of votes a candidate has

Name: the name of the candidates
Party: the name of the party this candidate belongs to.
Ballot: a list of ballots belongs to this candidate, only used for IR

Voting_System

Attribute name	Attribute type
votetype	string
totalballot	int
totalcandidate	int

Votetype: the type of vote, can be “OPL” or “IR”.
Totalballot: the total number of ballots in the file
Totalcandidate: the total number of candidates in the file

OPL_Voting_Sys

Attribute name	Attribute type
candidates	ArrayList<Candidate>
parties	ArrayList<Party>

Candidates: a list of candidates participate in this vote
Parties: a list of parties participate in this vote

IR_Voting_Sys

Attribute name	Attribute type
candidates	ArrayList<Candidate>
parties	ArrayList<Party>

Candidates: a list of candidates participate in this vote
Parties: a list of parties participate in this vote

IR_Ballot

Attribute name	Attribute type
voteRank	ArrayList<int>
currentRank	int

VoteRank: the order of ranks in this ballot
CurrentRank: the current rank of this ballot, starts from 1 and increases 1 for every redistribution.

4.2 Data Dictionary

See Section 3.2 and 4.1.

5. COMPONENT DESIGN

In this section, we take a closer look at what each component does in a more systematic way. If you gave a functional description in section 3.2, provide a summary of your algorithm for each function listed in 3.2 in procedural description language (PDL) or pseudocode. If you gave an OO description, summarize each object member function for all the objects listed in 3.2 in PDL or pseudocode. Describe any local data when necessary.

6. HUMAN INTERFACE DESIGN

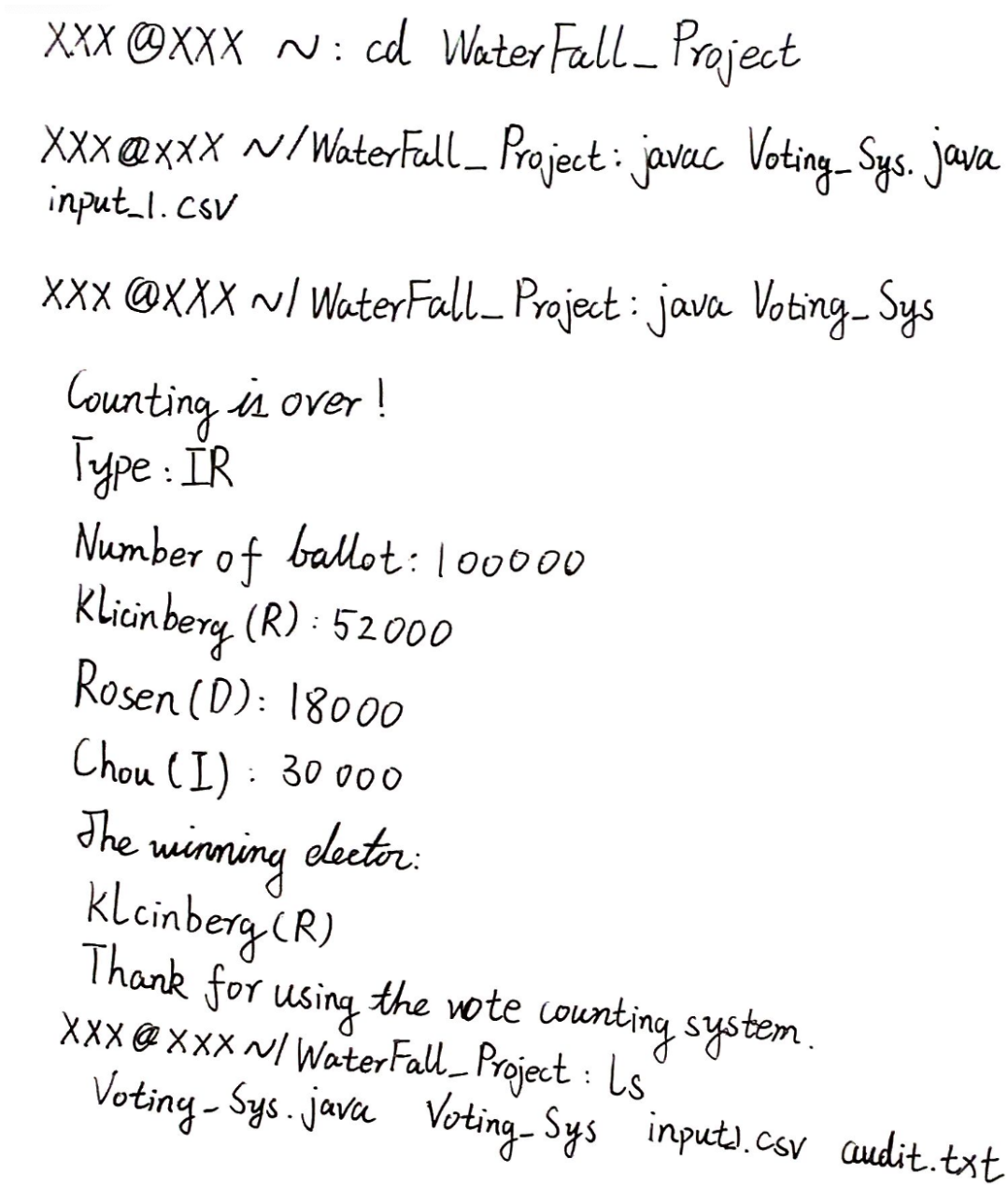
6.1 Overview of User Interface

Our system will have the following seven functionality.
Read: read the input file.
Decision: determine the vote type.
Process 1: File processing for Instant Runoff.
Process 2: File processing for Open Party Runoff.
Two-side Decision: Coin Flip simulation.
Display: Display the results on the screen for media.
Audit: Generalize the auditing file.

1. Users will provide the file of voting data as an input file. This will make the system satisfy the read functionality.
2. Users will choose a voting type in the provided input file. This will make the system complete the decision functionality.
3. The system will process the file based on the user defined requirements. (either process 1 IR or process 2 OPL)
4. The feedback information will be displayed on the screen. Also, users can access the information and procedure through the audit file. It records each step the system does to the data.

6.2 Screen Images

This program works for Windows, Mac and Linux, but we only provide the brief interface of the mac environment. Users need to adjust the command based on their system.



This briefly shows the interface of IR ballot. The example of the audit.txt file is provided in the SRS document section 3.1.3.

```
XXX@XXX ~/WaterFall_Project : javac Voting_Sys.java
input_2.csv
XXX@XXX ~/WaterFall_Project: java Voting_Sys
Counting is over!
Type: OPL
Number of Ballot: 100
Number of Seats: 4
Parties:
 D: 50%, 50%, 2 Seats
 R: 31%, 31%, 1 seats
 I: 19%, 19%, 1 seats
Candidates:
  Pike(D): 30
  Foster(D): 20
  Deutsch(R): 10
  Borg(R): 20
  Jones(R): 1
  Smith(I): 19
 The winning electors
  Pike(D)
  Foster(D)
  Borg(R)
  Smith(I)
Thank you for using the vote counting system.
```

This briefly shows the situation of the Open Party List ballot. The example of audit.txt file is provided in the SRS document section 3.1.3.

6.3 Screen Objects and Actions

On our screen, the only option is for the user to provide the input file. We don't have a GUI for users right now. There are no more actions. Details are listed in section 3.1 of the SRS document.

7. REQUIREMENTS MATRIX

Provide a cross reference that traces components and data structures to the requirements in your SRS document.

Use a tabular format to show which system components satisfy each of the functional requirements from the SRS. Refer to the functional requirements by the numbers/codes that you gave them in the SRS.

Use cases ID	**use cases name**	**Class**	**Functions**
VT_001	Read the input file	voting_system	readFile() getTotalballot() getgetTotalcandidate()
VT_002	Determine the vote type	voting_system	getVotetype()
VT_003	File processing for IR	IR_Ballot IR_Voting_sys	IR-Ballot() getRank() UpdateRank() Processfile()
VT_004	File processing for OPL	Candidate Party OPL_Voting_Sys	Candidate(string, Party) Candidate::getVote() Candidate::getName() Candidate::getParty() Party(string) Party:: getVote() Party:: getName() Party:: getMembers() Processfile()

VT_005	Coin flipping simulation	Coin_Flip	flip(int):int
VT_006	Display the result to the screen	IR_Voting_sys OPL_Voting_Sys	Processfile()
VT_007	Generate the Audit file	IR_Voting_sys OPL_Voting_Sys	Processfile()

8. APPENDICES

This section is optional.
Appendices may be included, either directly or by reference, to provide supporting details that could aid in the understanding of the Software Design Document.