Software Requirements Specification

for

Ballot Voting System

Version 1.0 approved

Prepared by Jan Achumbre (achum003), Andrew Tran (tran0904), Thien Nguyen (nguy3934), Jeffrey Chen (chen6377), Team 4

CSCI 5801

2/8/2023

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1. Introduction

1.1 Purpose

The purpose of this Software Requirements Specification (SRS) document is to clearly specify the requirements for the Ballot Voting System, Version 1.0. This document defines the functional and non-functional requirements of the system and serves as a contract between the stakeholders and the development team. The scope of the product covered by this SRS includes the complete functionality and features of the Ballot Voting System, Version 1.0…

1.2 Document Conventions

There are no outstanding typographical conventions followed in this document. For each title heading, information regarding it will be typed out in bold text directly below it. There is no system of priority in our documents which highlight or mark a certain requirement to have higher value over others.

1.3 Intended Audience and Reading Suggestions

Developers: The developers are expected to use this SRS to understand the requirements of the software being developed and the goals of the project. They will be able to use the information to build a solution that meets the project goals and expectations.

Users: The users will use this SRS to understand the features, capabilities, and functionalities of the software and how it will meet their needs.

Testers: The testers will use this SRS to understand the acceptance criteria for the software, including functional and non-functional requirements, and how the software will be tested.

Documentation Writers: The documentation writers will use this SRS to understand the software and its requirements to write clear and concise user documentation.

It is suggested that readers start by reading the introduction and the overview sections, which provide a high-level understanding of the project and its goals. Next, the reader can proceed to the sections that are most relevant to their role. For example, developers would focus on the functional requirements, while testers would focus on the acceptance criteria. The non-functional requirements and constraints and limitations sections can be read as needed to gain a complete understanding of the project requirements.

1.4 Product Scope

The scope of the product includes the implementation of two types of voting algorithms - Plurality/majority and Proportional representation, for the Waterfall project. The objective of the project is to provide a secure, user-friendly, and efficient platform for voting, and the goals include meeting the requirements of the stakeholders and delivering a high-quality product that satisfies the intended audience.

In terms of corporate goals and business strategies, this software aligns with the goal of delivering innovative solutions to help businesses succeed in a constantly changing environment. By leveraging the latest technologies and design principles, this platform is poised to set a new standard in business software.

1.5 References

FairVote.org. (n.d.). Party List Voting using Closed Party List (proportional voting type). [Online]. Available: https://www.fairvote.org/party_list_voting_using_closed_party_list

2. Overall Description

2.1 Product Perspective

This product is a standalone program and has no connection to any other existing database or other larger program. This product was assigned by Shana Watters, a University of Minnesota professor, as a project to handle the numerous voting systems in the United States. In its first iteration, this system will not act as a larger system to which it connects to a smaller subsystem that makes use of its results or functionalities.

2.2 Product Functions

- Instant Runoff Voting (IRV): Handle IRV, that is parsing through the received file and seeing which candidate has received the most votes; a majority.
- Create Audit File: After a file has been inputted, the program will create an audit file.
- Party List Voting using Closed Party Listing (CPL): Handles CPL by parsing through the received file and see which candidates won which seats.
- Tie: This function will handle the case when a tie occurs by using a coin flip virtually through a random functionality to decide which candidate or party wins and is elected into the house position.
- Displaying Result: Through the terminal, display the results of the election. What will be displayed will differ depending on which voting system is used. In the event that a

tie occurs, the system will explicitly state this and show which candidate won and show the difference in votes.

- Media Distribution: The program will create a file that lays out the results, which will be more detailed than the displayed result in the terminal. The file will be in the form of a standard txt file, but formatted such that the information is clear and easily understood.

2.3 User Classes and Characteristics

The main users of this product will be government or local officials who will be handling the election and its results. These people will use the product mainly through a User Interface such as a terminal where they are going to input a CSV file and get their results displayed. However, before the product is distributed, it will be used by testers to search for any bugs in the product, ensuring that they don't occur in the final product. Unlike the main users, these testers will have some access to the code, mostly the header files in order to facilitate gray box testing.

2.4 Operating Environment

Our project will be using C++. The current version of our compiler, which is g++ is 9.4.0 which is also the same for C++ as that is what is currently installed on the CSE Lab machines using the Linux Operating system. That being said, this should be able to run on any UofM CSE Lab machines. As for running the program, we will be providing a makefile in order to ensure the program dependencies are compiled properly. If we decide to distribute this program to other operating systems, then we will likely create a docker image of the program.

2.5 Design and Implementation Constraints

Limitations that we have are the type of file we can parse through (Strictly CSV files), Everything has to be done in C++, result file to be shared with the media might not be adequate or lack information, service(s) used to share the results might become unavailable in the future, and unable to change the format of the CSV file. Failure of following the design of the program could result in changes in outcome or failure to run the program.

2.6 User Documentation

- Program Manual (Tutorials Included)
- Sample result file that clearly labels type of information as well as how to read the format
- UML
- Use Cases
- This SRS document

2.7 Assumptions and Dependencies

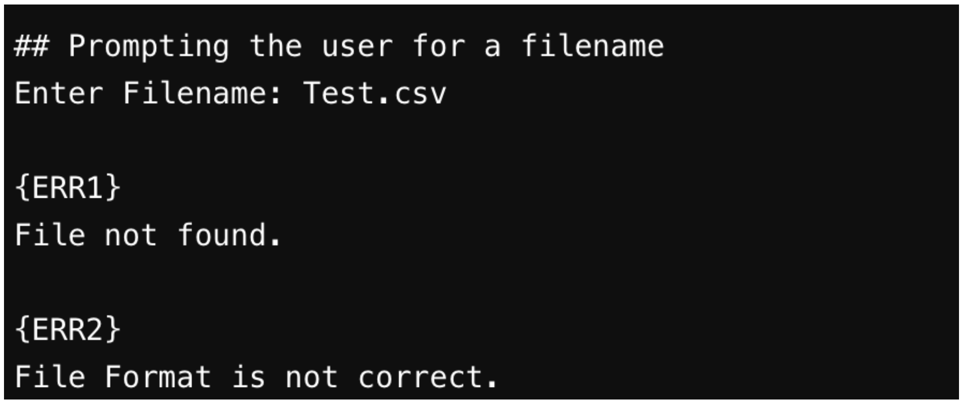
- **We can import existing libraries/modules**
- **We can format the result file in any way we want to**
- **The program will never be expected to communicate with a system or another program**
- **It is our choice whether or not we share our results with the media**
- **In the IR, if there is not a clear majority, then it will go into popularity votes.**
- **In the Closed Party Listing, if in the event that there is a supermajority with seats remaining, then the remaining seats will be put up for lottery.**

3. External Interface Requirements

3.1 User Interfaces

All of our user interactions will be through the Terminal, with no special shortcuts, and will be within the normal screen layout. Error Messages will be displayed in the terminal and will prompt the user to fix the problem through the terminal.

Below is a sample of what the user may see when entering the name of the CSV file. In this case, the user has already inputted “Test.csv”. Further below shows possible errors that might occur during this process and will lead to the user having to enter the file name once more.

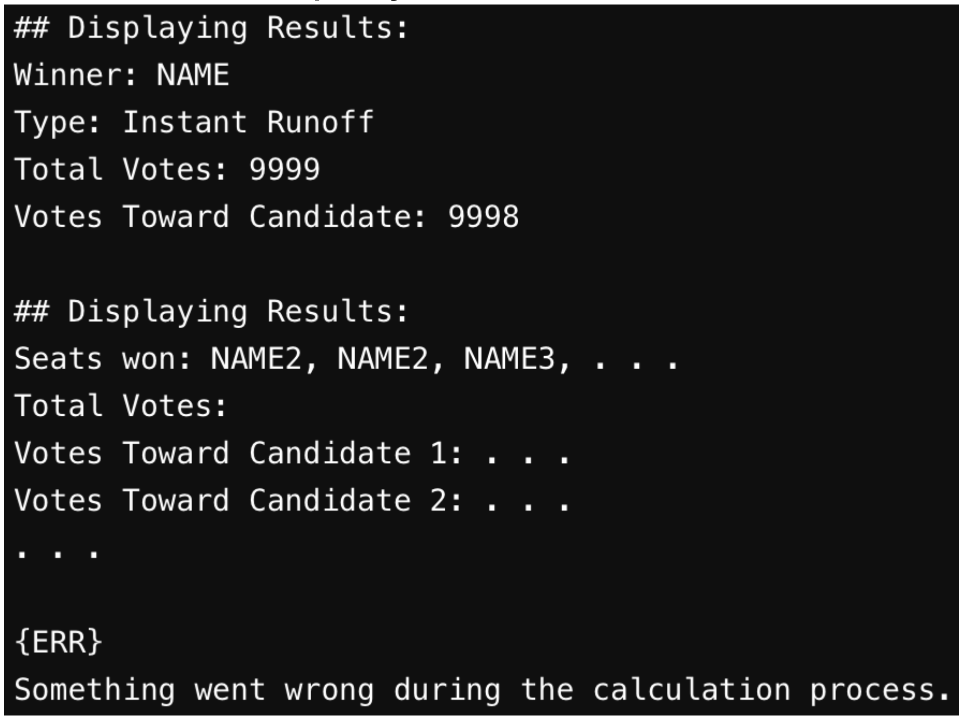


```
## Prompting the user for a filename
Enter Filename: Test.csv

{ERR1}
File not found.

{ERR2}
File Format is not correct.
```

This is another sample of what the user might see once the program has finished running. For both cases, each will have their own way of displaying results as can be observed in the image. These displays are subject to minor changes, especially in the cases of a tie which is not explicitly shown here.



```
## Displaying Results:
Winner: NAME
Type: Instant Runoff
Total Votes: 9999
Votes Toward Candidate: 9998

## Displaying Results:
Seats won: NAME2, NAME2, NAME3, . . .
Total Votes:
Votes Toward Candidate 1: . . .
Votes Toward Candidate 2: . . .
. . .

{ERR}
Something went wrong during the calculation process.
```

3.2 Hardware Interfaces

OS: Linux
CPU: Intel(R) Xeon(R) CPU E5-2699 v4 @ 2.20GHz (Anything considerably lower should be fine)
RAM: 4 gb
Network: Not required
Graphics: Not required

3.3 Software Interfaces

Our System will be based on C++ and so it will need to be installed on the system, the compiler that it will be run on will be g++ 9.4.0, These are on all of the UofM CSE Lab Machines and so additional information can be found on section 2.4 of this document.

There are no specific software components that the system must interface with. However, the system will need to be installed on a system that has the g++ 9.4.0 compiler installed and will require a terminal interface for user input and output.

The user will input a CSV file containing the voting data, and the system will parse and process the data to determine the election results. The results will then be displayed in the terminal and saved in an audit file. The system will also generate a media file containing

the results in a clear and easily understandable format that can be shared on multiple media platforms.

3.4 Communications Interfaces

Our program won't need to have an internet connection to run, but it is possible that an Internet connection might be needed to make sure that already installed components are up-to-date. Outside of this, our program will never make any network connections.

4. System Features

4.1 Reading the file

4.1.1 Description and Priority

The terminal asks the user input and the file should exist. Then the file should be read and able to proceed further. This feature is considered a High priority as it is crucial for the system to be able to accurately and efficiently process the election data in order to provide accurate results.

The cost of implementing this feature may be moderate, but the potential penalties of not having it would be significant as it would render the system useless.

4.1.2 Stimulus/Response Sequences

Terminal prompts the user for a filename. If not found, it prompts the user again else it will read in the file, open it and process it.

4.1.3 Functional Requirements

In order for this to work, the file should exist and should be formatted properly. The file also needs to be in CSV format and the first line in the file lists the type of voting.

REQ-1: UC_001
REQ-2: UC_009

4.2 IR

4.2.1 Description and Priority

If the first line of the input is IR that means it should be reading four lines of input. This is a high priority because it is one of the two important types of voting.

4.2.2 Stimulus/Response Sequences

1st Line: IR for instant runoff

2nd Line: The number of candidates.

3rd Line: The candidates are separated by commas.

4th Line: The number of ballots in the file.

4.2.3 Functional Requirements

In order for this to work, the file must be read, processed and the first line of the input has to say IR.

REQ-1: UC_004

4.3 CPL

4.3.1 Description and Priority

If the first line of the input is CPL that means it should be reading five lines of input. This is a high priority because it is one of the two important types of voting

4.3.2 Stimulus/Response Sequences

1st Line: CPL for closed party listing

2nd Line: The number of parties.

3rd Line: The parties separated by commas.

4th Line: The number of seats

5th Line: The number of ballots.

4.3.3 Functional Requirements

In order for this to work, the file must be read, processed and the first line of the input has to say CPL.

REQ-1: UC_007
REQ-2: UC_010
REQ-3: UC_011

4.4 Coin flip for tie

4.4.1 Description and Priority

If there is ever a tie, flip a coin. The system must randomly select the winner in a fair coin toss. If there is a super majority then the remaining seats, if any, become a lottery. This is a medium priority because this will apply only if a tie is detected between two or more candidates.

4.4.2 Stimulus/Response Sequences

The system detects a tie in the election and initiates a coin flip to determine the winner. The coin flip is fair and unbiased and the results of the coin flip are shown.

4.4.3 Functional Requirements

In order for this to proceed, there has been a tie between two candidates and the system has access to a random coin flip.

REQ-1: UC_005

4.5 Popularity for tie

4.5.1 Description and Priority

If there is not a clear majority in IR, then popularity wins after all votes have been handed out. This is a medium priority because this will apply if the majority isn't clear.

4.5.2 Stimulus/Response Sequences

The system calculates the results of the election or vote and determines a tie. It calculates the popularity of each candidate and determines the winner based on highest popularity.

4.5.3 Functional Requirements

In order for this to proceed, it needs to show that there is no majority.

REQ-1: UC_006

4.6 Displaying winner

4.6.1 Description and Priority

The winners of the election need to be displayed. This is a high priority since the purpose of the ballot processing is to find the winner of the election.

4.6.2 Stimulus/Response Sequences

The system receives the election results and processes the election results and calculates the number of ballots cast and the number of votes received by each candidate. Then it displays the winner of the election.

4.6.3 Functional Requirements

The file needs to be read and processed. The numbers need to be calculated in order to determine the winner.

REQ-1: UC_003
REQ-2: UC_008

4.7 Audit file

4.7.1 Description and Priority

The audit file should be produced with information about the election at that time. This is a medium priority because the audience cares more about the winner rather than the type of voting, number of candidates, etc.

4.7.2 Stimulus/Response Sequences

The system processes the election results and calculates the number of ballots cast and the number of votes received by each candidate. It displays the election results, including the number of ballots cast, the number of votes received by each candidate, the percentage of votes received, the winner(s), and any other relevant statistics.

4.7.3 Functional Requirements

The file needs to be read and processed and the information needs to be stored and kept tracked until it is written onto the audit file.

REQ-1: UC_002
REQ-2: UC_012
REQ-3: UC_013

4.8 Sharing with media personnel (Optional)

4.8.1 Description and Priority

The user will have the option to share the election results to the media. The results need to be in a suitable format in order to be able to export for media. This is low priority since it is the user's choice if they want to share it or not.

4.8.2 Stimulus/Response Sequences

The system provides the option to export the results for media. It asks the user for the format for the exported file. The user selects the format and the system exports the file.

4.8.3 Functional Requirements

The election has been conducted and the results are available in the system. This can allow the user to share the results.

REQ-1: UC_014

5. Other Nonfunctional Requirements

5.1 Performance Requirements

Our program will be run multiple times during the year at normal and special election times. During this time our program must be able to run 100,000 ballots in under 4 minutes for each election

Language: The system will be written in C++

Platform: The system will run on a CSE lab machine in Linux

Performance: An election should be able to run 100,000 ballots in under 4 minutes.

5.2 Safety Requirements

Our program doesn't have any safety requirements, since it will only calculate the winner of the election, there are no special safety or security requirements.

5.3 Security Requirements

Our program doesn't have any security requirements, allowing all of the users to have all of the same privileges. The security of ensuring one vote for one person is handled at the voting centers.

5.4 Software Quality Attributes

Correctness: The system should accurately calculate and display election results in a timely manner.

Usability: The system should be user-friendly and easy to use by ballot officials.

Availability: The system should be available for use by ballot officials during election periods.

Interoperability: The system should be able to work with different operating systems and be able to parse CSV files.

Reusability: The system should be designed in a modular way, making it possible for components to be reused in other projects.

Portability: The system (or product?) should be easily downloadable and come in a folder with its included files as stated 2.6 User Documentation.

Testability: The system should be able to be tested as gray or black box. The header files will be visible such that gray box testing can be properly facilitated.

Maintainability: The customers will NOT be able to maintain the product. If future developers require the need to make any changes, they will have to contact the product owners directly.

Adaptability: The system should be able to adapt to new technologies and changes in the environment.

Robustness: The system should be able to handle unexpected inputs, invalid data, and other exceptions without crashing or producing incorrect results.

5.5 Business Rules

The actual voting process will be done separately from the voting system. Ballots will be cast online and a comma-delimited text file will be provided to the system.

The program should prompt the user for any information that cannot be extracted from the file. However, the system should be designed to process the file for information where possible.

The system should be able to handle different types of voting, including Instant Runoff (IR) and Closed Party Listing (CPL). The type of voting will be specified on the first line of the file containing the ballots.

The system should not change the file structure outside of the program, as the election files will come in a predetermined format.

6. Other Requirements

<Define any other requirements not covered elsewhere in the SRS. This might include database requirements, internationalization requirements, legal requirements, reuse objectives for the project, and so on. Add any new sections that are pertinent to the project.>

Appendix A: Glossary

Instant Runoff Voting (IRV): A voting system used to determine a single winner from a list of candidates, where voters rank the candidates in order of preference.

Party List Voting using Closed Party Listing (CPL): A type of proportional representation voting system where voters vote for a political party rather than individual candidates.

SRS - software requirements specification

CSV: Comma-Separated Values

UI: User Interface

FTP: File Transfer Protocol

HTTP: Hypertext Transfer Protocol

UML: Unified Modeling Language, a standardized visual language used to model software systems.

Use Cases: A description of how users will interact with the Ballot Voting System to accomplish specific tasks.

Ballot Voting System: The software system being developed for the purpose of conducting elections.

Tie: A situation in which two or more candidates receive the same number of votes.

DELETE EVERYTHING BELOW LATER, INSERT USE CASES(?)

Appendix B: Analysis Models

TBD

Appendix C: To Be Determined List

<Collect a numbered list of the TBD (to be determined) references that remain in the SRS so they can be tracked to closure.>