# Software Requirements Specification

**for**

# *Trusty Vote Counter*

**Version 1.0 approved**

**Prepared by:**

**Alex Iliarski, iliar004, Team 16**

**Ethan Loukusa, louku008, Team 16**

**Gideon Tan, tan00160, Team 16**

**Robert Wang, wan00379, Team 16**

**University of Minnesota**

**February 16, 2023**

# Table of Contents

# Revision History

| Name | Date | Reason For Changes | Version |
|---|---|---|---|
| Initial commit | 2/16 | Uploading finished document | 1.0 |
| | | | |

# 1. Introduction

## 1.1 Purpose

The purpose of this document is to provide detailed specifications for *Trusty Vote Counter*. This software will process election data in the CSV file format and outputs the winner of the election and an audit file detailing the election results. This document describes the features and functionality of the system while also explaining the constraints of the software. Additionally, the document will explain how to interact with the system and how it will handle external input. This document is intended for users (e.g. election officials), developers, and testers of the system.

## 1.2 Document Conventions

This document was created based on the IEEE template for the System Requirements Specification Document.

## 1.3 Intended Audience and Reading Suggestions

Programmers and developers working on this product should be reading this document in its entirety from the beginning to the end in order to get the best idea on what the product is supposed to do as well as the use cases that are essential to the operation of the program.

Election Officials and any other users of this product should read Appendix A first and refer to those terms in case of any unknown references within the document. After reviewing the glossary, users should briefly review what this product does by reading the introduction section and the overall description. Lastly, the use cases will be the most important source of instructions on how to use the product and what certain functions are included in the program.

Testers should also get a basic understanding of the functions of this product by reviewing the introduction and overall description of the product. Besides this, they should mainly spend time examining the use cases section in order to have a basic understanding of what kinds of triggers should cause certain events.

*See Appendix A: Glossary for clarification on terms and abbreviations used in the document.*

## 1.4 Product Scope

This software is used to handle tabulation of ballots and determining winners in an election. The system can handle Instant Runoff (IR) and Closed Party List (CPL) elections. Given an input file with all the collected ballots from the election, the system will automatically output a winner to the screen and an audit file with specific data from the election. This ensures that there is no manual intervention necessary to determine a winner, ensuring correct results and limiting opportunities for fraud or user error. This allows for a quick and secure ballot counting process, ensuring that voters have a greater trust in the accuracy of the election results.

## 1.5 References

IEEE Template for System Requirements Specification Documents:
The Institute of Electrical and Electronics Engineers, Inc. Version 2022
https://goo.gl/nsUFwy

*Trusty Vote Counter* Use Case Document Version 1:
Authors: Robert Wang, Alex Iliarski, Ethan Loukusa, Gideon Tan (February 16, 2023)
https://docs.google.com/document/d/121MlFRjNwCR7pIU8JGo8Llk4MfVlZjnoYYk0-zYPD24/edit?usp=sharing

# 2. Overall Description

## 2.1 Product Perspective

*Trusty Vote Counter* is a standalone product designed for election officials to use to tally ballots for Instant Runoff and Closed Party elections automatically rather than manually. It is designed to handle CSV files and will display results to the system on which the election official is running the program, as well as to the media (if wanted). *Trusty Vote Counter* is intended to be incorporated into an online interactive voting system which can record the votes, put them into a CSV file, and run this program with the CSV file as input.

*Trusty Vote Counter* is intended to be used multiple times a year. This includes regular elections and special elections.

## 2.2 Product Functions

*This section describes the basic product functions of Trusty Vote Counter. See section 4 for more detailed documentation of these product functions.*

Input
- Identify target CSV file
- Determine necessary information about the election from the header
    - Type of election - IR or CPL
    - Other information pertaining to the election contained in the header
- Process file for an IR election
- Process file for a CPL election
- Prompt the user for any necessary information not included in the file
- Shuffle ballots to ensure election integrity

Determine the winner of an election
- Determine the winner of an IR election
    - If there is no majority and only two candidates remain, the winner is determined based on popularity.
    - In the case of a tie, randomly determine the winner.
- Determine the winner of a Closed-Party election.
    - In the case of a tie, randomly determine the winners and losers.

Export results of election
- Create an audit file
- Display results of the election to display

## 2.3 User Classes and Characteristics

The user classes that are expected to use this product the most are election officials, program developers, and testers. The election officials are the customers that will be putting this program into practical use when they are trying to use the program to decide the winner of an election. These officials can be expected to have experience in how the types of elections work and will probably know how to use the program, however they will have a more difficult time understanding how the code works internally. The developers will be the individuals who are tasked with laying out the groundwork for the requirements and actually building the software itself. This group will have a very good understanding of how the internals of the code works and can fix any bugs that may arise in the process. Lastly, the testers are expected to test the code for bugs and report them back to the developers. These testers may consist of developers as well as people outside of development, which will create a more robust environment to test the program in.

## 2.4 Operating Environment

This software will run on any Java Runtime Environment (JRE), regardless of the user's operating system. The JRE should be running Java version 17.0.5. This code will be developed and tested inside an Integrated Development Environment (IDE), and run directly inside the terminal. The software must be able to run on any CSELabs machine, which all use Java version 17.0.5.

## 2.5 Design and Implementation Constraints

The limits in our flexibility to develop this software are few in number and contain the basics of what makes a program good and functional. The largest constraint is that the program should only be written in the Java programming language, and with that comes the specific organization of the files in accordance with Java's file access system. Beyond this, concerns include the efficiency of the program, organization so that it is readable to developers and testers, and a simple text-based interface that the user can easily interact with without necessarily having to understand the code.

## 2.6 User Documentation

The main source of information regarding the functionality of this project will be detailed in this document. Linked in the references (*1.5*) is the use cases document, where the main functions of the program will be specified, with trigger conditions and preconditions will be made available to the user. Beyond this, inside the code, other developers, or the users themselves can expect to see comments documenting certain areas of the code in order to make all the code more readable.

## 2.7 Assumptions and Dependencies

The dependencies of this project incorporate a large part of the software that the program was built using. Some of the main software that this project relies on are the Java programming language compiler to run and test the code that we will be writing, the Integrated Development Environment that the developers will be using, and Git/Github to keep track of the code version histories and allow the team to work together.

Since this code is largely expected to run inside of an IDE, the user would be expected to download the code onto their own system to run inside a terminal.

# 3. External Interface Requirements

## 3.1 User Interfaces

The main user interface for this system will be in the form of a terminal inside the development software that the developers use to write and test the code. This interface will essentially just consist of text-based commands and results that are printed back into the terminal for the user to see. In order to make the data more readable both for the developers and the users, the text data will be arranged into tables with tables and columns. Because this program runs directly on the software that it is being developed in, the user will be expected to download the program onto their system and use a terminal to run it.

## 3.2 Hardware Interfaces

There are anticipated to be a few specialized hardware interfaces that are crucial to the operation of this product. Essentially all that is required to run this program would be any modern computing machine with a display and a method of inputting text like a keyboard and mouse. Inputting text will essentially be the way the user interacts and inputs information into the program for it to run with certain files. The monitor will be responsible for displaying the results of the program after running it over the sets of data that the user inputs.

## 3.3 Software Interfaces

First and foremost, this product will be developed in Java, so downloading Java and any libraries used will be necessary to use *Trusty Vote Counter*. One of the most important software interfaces that this project depends on is the IDE that the code for the program is developed on. The code will be expected to be run directly inside any IDE and it is there that the results for the program will be printed for the user to see. Besides the IDE itself, the development and maintenance of the product will also be using Git v2.39.2 and GitHub at github.com as a means to version control the product and to allow the development team to work together efficiently. Outside of the most important components above, it is also required that within the Java package, the files that contain the information about the ballots are recorded in a format that is readable and accessible for the user and the system.

Additionally, there are some inputs and outputs to consider. An input CSV file holds the necessary data to determine the winner of the election. This file will be input either through the command line or upon program execution. This file must be in the same directory as the program. Another input to the program is user-entered information about the election. If necessary to prompt the user for information not found in the input file, the user will enter text in the command line that will be used as important election information to proceed with the necessary algorithms. As output, at the end of the program's execution, an audit file is created in the same directory as the program. This audit file contains details of the election process, such as the redistribution of ballots in IR voting and the number of seats allocated to each party in CPL voting.

## 3.4 Communications Interfaces

No communication with outside systems is necessary. However, in order to run the *Trusty Vote Counter* program, there must be one and only one CSV file containing ballot information in the same directory, the data on which the program will perform the counting.

# 4. System Features

## 4.1 Identify and open the file that contains ballots

| | |
|---|---|
| Name | Identify and open the file that contains ballots |
| ID | UC_001 |
| Description | Retrieve the file information from the ballots, from the user to run election algorithms on. The file name is retrieved via command line argument or when the user is prompted. |
| Actors | Election Official, Developers, Testers |
| Organizational Benefits | Simplifies and streamlines the process of identifying the election file. |
| Frequency of Use | ● Once per election for election official<br>● Many times for developers and testers |
| Triggers | The program has started. |
| Preconditions | ● File must be in the same directory as the program.<br>● There are no errors in the file. |
| Postconditions | The file is accessed and opened. |
| Main Course | 1. The name of the file is retrieved from the command line argument.<br>2. An attempt is made to open the file. |
| Alternate Courses | 1. The user is prompted for the name of the file.<br>2. An attempt is made to open the file. |
| Exceptions | ● File is written in an invalid format.<br>● Insufficient read permissions on the file.<br>● File is not in the same directory.<br>● The user abandons the current session and does not respond to the prompt. |

REQ-1: Input file should be a correctly formatted .csv file (see *4.3.2*/*4.4.2*) in the same directory as the program.

REQ-2: If the file is not found or it is incorrectly formatted, an error will be thrown.

## 4.2 Determine the election type based on the first line of the input file

| | |
|---|---|
| Name | Determine the type of election based on the first line of the file |

| | |
|---|---|
| ID | UC_002 |
| Description | Reads the first line of the file and determines whether the file contains the election information for a CPL election or an IR election. |
| Actors | System |
| Organizational Benefits | Quickly determines whether to follow a CPL ballot extraction flow or an IR ballot extraction flow. |
| Frequency of Use | Once per election. |
| Triggers | Election file is opened. |
| Preconditions | ● Election file is opened successfully with no errors or exceptions.<br>● First line of the file is either “CPL” or “IR”. |
| Postconditions | The type of election is known, and the appropriate election can be run on the information in the rest of the file. |
| Main Course | 1. The first line is read.<br>2. The type of election is determined based on the information on the first line.<br>3. If the first line is “IR”, see use case UC_003. If the first line is “CPL”, see use case UC_004. |
| Alternate Courses | None |
| Exceptions | ● File was not opened successfully<br>● First line does not say “IR” or “CPL” |

## 4.3 Process file with IR election information

| | |
|---|---|
| Name | Process file with IR election information |
| ID | UC_003 |
| Description | Process the information in the CSV file for an IR election. |
| Actors | System |
| Organizational Benefits | Allows all the election information to be stored in the program, so the file is only read once. |
| Frequency of Use | Every time an IR election is run (max once per time program is run). |
| Triggers | It is determined that the election is IR, and all the information has been determined. |

| | |
|---|---|
| Preconditions | ● The file is opened and valid.<br>● The file contains information about an IR election. |
| Postconditions | ● The ballots and information about the ballots have been read and processed.<br>● The file is closed. |
| Main Course | 1. The number of candidates is read from the file and stored in the program.<br>2. The name of each candidate is read from the file and stored in the program.<br>3. The number of ballots is read from the file and stored in the program.<br>4. Each ballot is read from the file and stored in the program<br>5. The file is closed. |
| Alternate Courses | None |
| Exceptions | Improperly formatted CSV file. |

*4.3.2 CSV format for IR elections*

As previously mentioned, the first line denotes the type of election (IR). The second line denotes the number of candidates. The third line lists the candidates, separated by commas. The fourth line denotes the total number of ballots cast in the election. The following lines are the ballots themselves, where, for example, a 1 in the second position would denote the second candidate listed on the third line being that voter's number one choice. It is assumed that the file is correctly formatted and that each ballot ranks at least one candidate, though not necessarily all of the candidates.

Note: *Trusty Vote Counter* does not currently support write-in candidates for IR elections.

## 4.4 Process file with CPL election information

| | |
|---|---|
| Name | Process file with CPL election information |
| ID | UC_004 |
| Description | Process the information in the file for a CPL election. |
| Actors | System |
| Organizational Benefits | Allows all the election information to be stored in the program, so the file is only read once |
| Frequency of Use | Every time a CPL election is run (max once per time program is run). |
| Triggers | It is determined that the election is CPL, and all the information has been determined. |

| | |
|---|---|
| Preconditions | ● The file is opened and valid.<br>● The file contains information about a CPL election. |
| Postconditions | ● The ballots and information about the ballots have been processed.<br>● The file is closed. |
| Main Course | 1. The number of parties is read from the file and stored in the program.<br>2. The name of each party is read from the file and stored in the program.<br>3. The name of each candidate for each party is read from the file and stored in the program. Each candidate is associated with their respective parties.<br>4. Any independent candidate is considered their own party, and shall be the only candidate in their party.<br>5. The number of seats is read from the file and stored in the program.<br>6. The number of ballots is read from the file and stored in the program.<br>7. Each ballot is read from the file and stored in the program.<br>8. The file is closed. |
| Alternate Courses | None |
| Exceptions | File is not properly formatted. |

*4.4.2. CSV format for CPL elections*

As previously mentioned, the first line denotes that this is the election information for a CPL election. The second line denotes the number of parties. The third line lists the parties, separated by commas. Immediately after the third line, each line is the candidates for each party. For example, the second line after the third line (line 5) corresponds to the second party listed on line 3. The candidates on line 5 are the ranked order of candidates for that party. Immediately after the candidates lines, a line denotes the number of seats to be allocated this election. Immediately after the seats line, a line denotes the number of ballots cast in this election. The rest of the lines are the actual ballots cast. The position of the 1 in the line corresponds to a vote for the party in the same position on line 3.

Note: *Trusty Vote Counter* does not currently support write-in candidates for IR elections.

## 4.5 Determine a winner in IR

| | |
|---|---|
| Name | Determine a winner in IR |
| ID | UC_005 |
| Description | Tallies all the first choice votes for each candidate, as well as the votes of eliminated candidates. If a candidate has a majority, he is declared the winner. If no majority is found and candidates can still be eliminated, a candidate is eliminated, and votes are reallocated(see use case UC_006). If no majority is found after all possible ballot redistribution, then the plurality |

| | |
|---|---|
| | vote is used(see use case UC_007). |
| Actors | System |
| Organizational Benefits | Provides a way to determine the result of an IR election |
| Frequency of Use | Once per IR election |
| Triggers | ● IR election results are input and read |
| Preconditions | No winner has been determined in an IR election |
| Postconditions | A winner has been determined |
| Main Course | 1. The first-choice ballots are tallied<br>2. Determine the candidate with the most votes. If this candidate has over 50% of the votes as the winner, they are declared to be the winner.<br>3. If there is no candidate with a majority, the lowest ranked candidate is eliminated and their ballots are redistributed according to the voter's next choice. If there is a tie for the lowest scoring candidates, a loser is determined by a random process(See UC_010).<br>4. Repeat steps 2-4 until a winner is determined.<br>5. If all ballots have been redistributed and no majority candidate has occurred, a plurality vote is used to determine the winner. See use case UC_007 for details. |
| Alternate Courses | ● In step 2, if no candidate has the majority, a candidate is eliminated and the ballots are retallied (see UC_006)<br>● If votes can no longer be reallocated or there are no more candidates to be eliminated, the candidate with the most votes is declared the winner (see UC_007)<br>● If there is a tie among the lowest ranking candidates, a loser must be determined. See use case UC_010 for details. |
| Exceptions | None |

4.5.2 *Clarification on redistribution of votes*

To clarify, the way that the lowest scoring candidates are redistributed is that those voters whose primary votes are for the losing candidate now vote for their next choice. E.g. Alice's first choice is Bob, but Bob is eliminated so his votes are reassigned. Alice's vote now counts towards Billy, her next choice. If Alice has no next choice, her ballot is effectively thrown away. See section 4.6 below for more information.

## 4.6 Eliminate a candidate and reallocate the votes (IR)

| | |
|---|---|
| Name | Eliminate a candidate and reallocate their votes |
| ID | UC_006 |
| Description | In the case that there is no majority winner in the IR voting process, the system should reallocate the votes of the candidate that placed last until a winner can be decided. |
| Actors | System |
| Organizational Benefits | Correctly and automatically performs the instant runoff voting algorithm as required. Automates the reallocation of votes, reducing the need for manual intervention. |
| Frequency of Use | The system should reallocate the votes every time a candidate does not get the majority of the votes after a round of the IR voting process. This can happen 0 to (n-1) times, where n is the number of candidates in the election. |
| Triggers | Reallocation should trigger if there is no candidate with over 50% of the votes in an IR system after a round of counting the votes, unless there are no ballots left to redistribute. |
| Preconditions | The system should have been running the IR vote counting algorithm, and there is no winner after the current round of counting the votes, unless there are no ballots left to redistribute. In the case that there is a tie between the lowest-placing candidates, see use case UC_010 for how a fair coin toss is used to determine the loser. |
| Postconditions | ● The candidate in last place is no longer being considered<br>● All the ballots that were placed in favor of the last place candidate are allocated in favor of the next candidate indicated on the ballot. |
| Main Course | 1. The candidate with the least votes is determined<br>2. The list of ballots is traversed. For each ballot, if the vote was given to the candidate with the least votes, the vote is given to the next choice indicated on the ballot. |
| Alternate Courses | ● In step 1, if two or more candidates are tied, a loser is randomly determined(See UC_010). |
| Exceptions | None |

## 4.7 Determine a winner by plurality (IR)

| | |
|---|---|
| Name | Determine a winner by plurality (IR) |
| ID | UC_007 |
| Description | If all ballots have been redistributed to the fullest extent, and there is still no |

| | |
|---|---|
| | candidate that holds a majority in the IR voting system, the candidate with the plurality of the votes is deemed to be the winner. |
| Actors | System |
| Organizational Benefits | Accounts for the possibility of being unable to determine a winner due to a majority not being found and no other ways to proceed. |
| Frequency of Use | Possible in every IR election |
| Triggers | Ballots are tallied for each candidate |
| Preconditions | ● No other candidates can be eliminated<br>● Ballots can no longer be retallied without resulting in the same outcome |
| Postconditions | The winner of the IR election is determined |
| Main Course | 1. Determine that no other candidate can be eliminated, either because there is only one candidate left, or because votes can no longer be reallocated.<br>2. Recount all ballots(including those for candidates who were previously eliminated).<br>3. Choose the candidate with the most votes as the winner. If there is a tie for the highest-scoring candidates, see use case UC_010 for determining the winner. |
| Alternate Courses | ● In step 3, if two or more candidates are tied, a winner is randomly determined (See UC_010). |
| Exceptions | None |

## 4.8 Determine the number of seats for each party in a CPL election

| | |
|---|---|
| Name | Determine the number of seats for each party in a CPL election |
| ID | UC_008 |
| Description | Tallies all the votes for each party, and uses the largest remainder algorithm to determine how many seats each party should get. |
| Actors | System |
| Organizational Benefits | Provides an efficient and reliable way to determine the results of a CPL election. Additionally, the lack of manual intervention by election officials allows for more trusted results. |
| Frequency of Use | Once in every CPL election. |

| | |
|---|---|
| Triggers | CPL election ballots are input and processed. |
| Preconditions | No winner has been determined in a CPL election |
| Postconditions | A winner has been determined for the election |
| Main Course | 1. Votes for each candidate are tallied.<br>2. In the case that there is a tie between parties, see use case UC_009. This fully determines the ordering of the parties.<br>3. Using the largest remainder algorithm, determine the number of seats that each party gets.<br>4. Seats are fully allocated and the process of outputting the winner and audit file can proceed. |
| Alternate Courses | ● In step 3, if two or more candidates are tied, a loser is randomly determined(See UC_010).<br>● In step 3, If a party has more seats allocated to it than they have candidates, seats are reallocated(See use case UC_009). |
| Exceptions | N/A |

*4.8.2. Description of largest remainder algorithm*

*Note: Further information on this algorithm can be found on FairVote.org*

1. The first seats are allocated via a quota, which is calculated by dividing the total number of votes and dividing by the number of seats. Any party with that amount of votes gets a seat per amount of times they achieve the quota. The remaining votes become that party's remainder.
    a. E.g. Party A receives 35,000 votes and the quota is 10,000 votes/seat. Party A receives 3 seats and has a remainder of 5,000.
2. The remaining seats go to the parties with the highest remainder. E.g. If there are three remaining seats, they go to the three parties with the highest remainders.
3. If a party is allocated more seats than they have candidates listed, a process is used to reallocate these seats. See 4.9 for details.
4. If there is a tie between parties for their remainder, a fair tie-breaking process is used. See 4.10 for details.

## 4.9 Reallocate seats if needed (CPL)

| | |
|---|---|
| Name | Reallocate seats if needed (CPL) |
| ID | UC_009 |
| Description | In closed party list voting, a party can receive so many votes such that they are allotted more seats than they have candidates listed. This is handled by recalculating the number of seats for each party after eliminating the particular party with not enough candidates, and reallocating the remaining |

| | |
|---|---|
| | seats. |
| Actors | Election officials, developers, and testers |
| Organizational Benefits | Correctly and automatically resolves the problem of a party being allocated more seats than there are candidates. This ensures that algorithms are correct and do not require manual intervention. Hence, this ensures trusted and correct voting systems. |
| Frequency of Use | It is possible that this could occur in every closed party election. |
| Triggers | The number of seats for each party is determined |
| Preconditions | One or more parties have been allocated more seats than they have candidates |
| Postconditions | Seats have been redistributed such that all parties are allocated a number of seats less than or equal to the number of available candidates |
| Main Course | 1. Allocate the party with not enough candidates the number of seats equal to the number of their candidates.<br>2. Use the largest remainder algorithm on the remaining seats and parties to determine the number of seats each remaining party should have. |
| Alternate Courses | Repeat steps if necessary because there are other parties with more seats than candidates. |
| Exceptions | N/A |

## 4.10 Determine a winner or loser in case of a tie

| | |
|---|---|
| Name | Determine a winner or loser in case of a tie |
| ID | UC_010 |
| Description | In the case that there is a tie in any voting system between two or more candidates, one candidate is chosen as the winner or loser, depending on the scenario. |
| Actors | System |
| Organizational Benefits | Allows for tied rounds to be automated and not require manual intervention by an election official. Instills trust and integrity in the system by ensuring fair resolution to ties. |
| Frequency of Use | It is possible to occur every single time an election is run, although predicted to be fairly infrequent. |
| Triggers | ● There is a tie between the lowest-scoring candidates in a particular |

| | |
|---|---|
| | round of IR voting, and a loser needs to be determined.<br>● There is a tie between the highest-scoring candidates in IR voting when there are no more ballots to redistribute, and a winner needs to be determined.<br>● There is a tie between parties in CPL, and a candidate needs to be allocated a seat. |
| Preconditions | ● File has been successfully input and read.<br>● Tabulation of ballots has concluded.<br>● There is determined to be a "significant" tie. |
| Postconditions | The ordering between the tied candidates/parties has been determined in an unbiased manner and the algorithms can proceed as necessary. |
| Main Course | 1. A tie for the winning candidate occurs.<br>2. A candidate is randomly chosen out of all the tied candidates.<br>3. The chosen candidate is declared the winner. |
| Alternate Courses | 1. A tie for the losing candidate occurs.<br>2. A candidate is randomly chosen out of all the tied candidates.<br>3. The chosen candidate is declared the loser, and the process proceeds from there. |
| Exceptions | None |

*4.10.2. Description of tie-breaking algorithm*

The tie is broken through a "random pool" process. In this process, a random number between 0 and 1 is first generated 1000 times to ensure true randomness, and then another random number is generated as the 'target' value. Each candidate/party involved in the tie-breaking process will also have a random number generated for them. These candidates/parties are then ordered by the absolute value of the difference between their number and the target number. The smallest difference is considered to be a winner, while the largest distance is the loser. This new official ordering is then used for whatever tied situation needs to be resolved.

## 4.11 Shuffle Ballots

| Name | Shuffle Ballots |
|---|---|
| ID | UC_011 |
| Description | The ballots should be randomized in their order upon the start of the program to ensure that the election is being held as fairly as computationally possible. |
| Actors | System |
| Organizational Benefits | Helps to reduce bias in an election by randomizing the order of the ballots because certain ordered ballots could result in a different winner than another order. |

| | |
|---|---|
| Frequency of Use | Shuffling should be used before any iteration of the program that counts the votes. |
| Triggers | Starting the program should shuffle all the input ballot information as one of its first few steps. |
| Preconditions | The program should be running and files containing the ballot information should be present in the system. |
| Postconditions | The organization of ballots inside the system should now be randomized. |
| Main Course | 1. Files are being inputted into the system.<br>2. The system read the files when the program first starts to run.<br>3. After all the election information is stored, the program should now shuffle all of the ballots.<br>4. After the previous step, the program is now ready to start the actual process of counting the votes. |
| Alternate Courses | N/A |
| Exceptions | The ballots should only need to be shuffled once or twice, so after that set amount, the system should move on and so it should not be running the shuffle method anymore. |

## 4.12 Creating audit file

| | |
|---|---|
| Name | Creating audit file |
| ID | UC_012 |
| Description | Produces an audit file in text form, which contains the election information: The election results, how the election progressed, who got which ballots, and the order of ballots being received (if applicable). This information should suffice if it is deemed necessary to reproduce the election. |
| Actors | Election Official, Developers, Testers |
| Organizational Benefits | Produces a file with all necessary election information. This information shows how the election progressed at each step. Should also show the order of candidate removal and ballot redistribution in the case of an IR election. Allows for election results to be reproducible and understandable. |
| Frequency of Use | ● Once per election for election official<br>● Many times for developers and testers |
| Triggers | Election results are determined. |
| Preconditions | Election results are valid. |

| | |
|---|---|
| Postconditions | Audit file exists in the same directory and election results are stored there. |
| Main Course | 1. Information from election results is extracted.<br>2. Information is written to a new file in the same directory. |
| Alternate Courses | In step 2, if there is a previous file with the same name, it is overwritten(See UC_013). |
| Exceptions | The file was unable to be created. |

## 4.13 Previous audit file is overwritten

| | |
|---|---|
| Name | Previous audit file is overwritten |
| ID | UC_013 |
| Description | All iterations of the audit file that the program outputs should be named the same in order for future calls to the program to overwrite past audit file information. In the case that an audit file with the same name already exists, it must be overwritten. |
| Actors | System |
| Organizational Benefits | ● Duplicate audit files of the same election are avoided<br>● Reduces the number of ballot files produced |
| Frequency of Use | Whenever the program is run on a previously run election file. |
| Triggers | The audit file is generated. |
| Preconditions | An audit file of the same name already exists in the same directory. |
| Postconditions | All previous audit files are overwritten with the current audit file. |
| Main Course | 1. An audit file is produced.<br>2. An audit file, which has the same name and already exists in the same directory, is overwritten. |
| Alternate Courses | None |
| Exceptions | ● The previous audit file could not be deleted.<br>● The new audit file was unable to be created. |

## 4.14 Display election results to the screen

| | |
|---|---|
| Name | Display election results to the screen |

| | |
|---|---|
| ID | UC_014 |
| Description | After the program is finished running, the name of the winning candidate, party, or parties, along with the number of ballots cast and the stats for each party or candidate, should be printed on screen for the user to see. |
| Actors | Election Officials, Developers, Testers |
| Organizational Benefits | Allows for easy viewing and recording of the winner of the election because the user would not have to look through the data to see the winner and other stats from the election. |
| Frequency of Use | After every completed run of the program, the display should print the winner on the screen. |
| Triggers | Election results are determined. |
| Preconditions | Election results are valid. |
| Postconditions | The information from the election is printed to the screen. |
| Main Course | IR:<br>1. The election results are fully determined.<br>2. The name of the winning candidate is outputted to the terminal (IR), along with the type of election and stats about the election. |
| Alternate Courses | CPL:<br>1. The election results are fully determined.<br>2. The winners of the seats are outputted to the terminal, along with the type of election, number of seats, and stats from the election. |
| Exceptions | None |

## 4.15 Share election results with media

| | |
|---|---|
| Name | Share results of the election with media |
| ID | UC_015 |
| Description | The results of the election will be shared with media members via email. |
| Actors | Media members |
| Organizational Benefits | Allows for the results of the election to be easily shared with the public. |
| Frequency of Use | 100% (Once every election) |
| Triggers | The completion of the counting of the votes and therefore declaration of the winner should trigger this. |

| | |
|---|---|
| Preconditions | ● The votes have all been counted.<br>● The proper media members are identified and their emails are accessible. |
| Postconditions | All of the necessary media members have the results of the election. |
| Main Course | 1. Put the results of the election into a file(.txt).<br>2. Email the file to all media members on the list. |
| Alternate Courses | 1. Construct an email where the results of the election are in the text of the email itself.<br>2. Send the email to all media members on the list. |
| Exceptions | No media members have requested the election results<br>The email fails to send |

## 4.16 User can be prompted for any missing information

| | |
|---|---|
| Name | The user is prompted for any missing information |
| ID | UC_016 |
| Description | In the case that some information cannot be parsed or is slow to parse from the given files, the user can be prompted to enter information about the given file. |
| Actors | Election Officials, Developers, Testers |
| Organizational Benefits | Allows for fast access to information about ballot information files being passed in. |
| Frequency of Use | The option should be available at the start of each run of the program, but the case that some information will not be available inside a given file will probably be less than half the time. |
| Triggers | The start of the program should trigger a prompt for extra information about the file. At the end of parsing the given file, if some information cannot be found, should prompt the user for information again. |
| Preconditions | The program should already be running and the terminal should be open for test interaction between the system and the user |
| Postconditions | After all the information is entered by the user, the required information about the ballots being passed in to determine the winner should all be available |
| Main Course | 1. The program is run by the user<br>2. The file name containing the ballot information should be entered to the system along with prompts asking for more information upfront.<br>3. The file is read by the system and information within is parsed<br>4. The user is again asked for any information not readily made available in the file |

| | |
|---|---|
| Alternate Courses | None |
| Exceptions | No extra information is needed from the user |

# 5. Other Nonfunctional Requirements

## 5.1 Performance Requirements

The program is required to handle 100,000 ballots in under 4 minutes. This constraint is in place so that the winner of an election can be determined in a reasonable time frame and announced to the public accordingly. Quick determination of election results helps to ensure the public maintains trust in the results of the election process.

## 5.2 Safety Requirements

Although the input file is only being read by the software, not written to, it is possible that the file could become corrupted if, for example, the computer is shut down while the file is being read. As such, make sure to always allow the execution of the program to finish before shutting down the computer. Additionally, it is always good practice to keep a backup copy of the file stored somewhere, just in case the file becomes corrupted during the execution of the program, although this is not anticipated.

## 5.3 Security Requirements

There are no special security requirements since security is handled during the voting process itself, which is outside the scope of the *Trusty Vote Counter*.

## 5.4 Software Quality Attributes

Some of the most important quality characteristics of this product are its correctness, maintainability, portability, and extensibility. Since we already know what kinds of elections we will be dealing with and the steps it takes in determining a winner, correctness will be something that this product adheres to strongly. Also, when it comes to maintaining this program over an extended period of time, a GitHub repository will be used to store the code and to keep track of any past and future changes in case there comes a need to reference previous versions. We are additionally anticipating this product to be a very lightweight yet robust program that will be easily shared, learned, and implemented in various different situations as the users deem necessary. Lastly, the design of this product in an object-oriented programming language allows for easy extensions in the future in the possible case that the user might eventually want to be able to enter more types of files or election types into the program.

## 5.5 Business Rules

There are not any strict business rules for this product because it is built in a way that is intended to be used in the same way by all the users. In the same way that there is not too much of a security or role

verification requirement for this product, since the security takes place during the actual voting process itself, there are no functional requirements to enforce the rules. Perhaps the only concern, in this case, would be inputting invalid file/text information into the terminal when running the application. In this case, there would be certain syntax rules to adhere to, but are not limited to certain individuals or roles.

# 6. Other Requirements

# Appendix A: Glossary

**CPL** - Acronym for "Closed Party List" voting system, which is a voting system in which each political party has pre-decided who will receive the seats allocated to that party in the elections, the winner will be decided based on these orderings. Voters vote for a preferred party, and the percentage of votes that each party receives, along with the greatest remainder algorithm, determines the number of seats allocated to each party.

**CSV** - Acronym for "Comma Separated Values", which is a file format where different columns of a table are represented as a list of values with commas in between each value and the rows represent the different rows in such a table.

**Git** - Git is a "version control" software that is used to help manage the changes inside a document that helps a team of developers keep track of any changes and work on the same file effectively.

**GitHub** - Github is a website that uses "Git" to store "repositories" which contain the programs that developers are using, and allows the team to maintain a main file where production-ready code is stored

**IDE** - Acronym for "Integrated Development Environment", which is the software that is used to write, compile, and or interpret the code that developers write.

**IR** - Acronym for "Instant Runoff" voting system, which is a type of ranked preferential voting method. It uses a majority voting rule in single-winner elections where there are more than two candidates.

**Terminal** - A terminal is simply a text-based interface to the computer. In a terminal, you can type commands, manipulate files, execute programs, and open documents.

**Version Control** - This refers to the ability to keep track of changes to a certain document and allows ease of access to past versions of documents.

# Appendix B: Analysis Models

No pertinent analysis models are included in this product.

# Appendix C: To Be Determined List

No references to be determined.