**Software Requirements Specification**

**for**

**Voting System**

**Version ##**

**Prepared by:**

Sami Frank (fran0942)

Jake Waro (warox001)

Declan Buhrsmith (buhrs001

Allison Miller (mill7079)

**University of Minnesota- Twin Cities: CSCI 5801**

**2.12.2020**

**Table of Contents**

**Table of Contents ii**

**Revision History ii**

**1. Introduction 1**

1.1 Purpose 1

1.2 Document Conventions 1

1.3 Intended Audience and Reading Suggestions 1

1.4 Product Scope 1

1.5 References 1

**2. Overall Description 2**

2.1 Product Perspective 2

2.2 Product Functions 2

2.3 User Classes and Characteristics 2

2.4 Operating Environment 2

2.5 Design and Implementation Constraints 2

2.6 User Documentation 2

2.7 Assumptions and Dependencies 3

**3. External Interface Requirements 3**

3.1 User Interfaces 3

3.2 Hardware Interfaces 3

3.3 Software Interfaces 3

3.4 Communications Interfaces 3

**4. System Features 4**

4.1 Inputting file name(s) 4

4.2 Shuffle switch 4

4.3

4.4

4.5

4.6

**5. Other Nonfunctional Requirements 4**

5.1 Performance Requirements 4

5.2 Safety Requirements 5

5.3 Security Requirements 5

5.4 Software Quality Attributes 5

5.5 Business Rules 5

**6. Other Requirements 5**

**Appendix A: Glossary 5**

**Appendix B: Analysis Models 5**

**Appendix C: To Be Determined List 6**

**Revision History**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Date** | **Reason For Changes** | **Version** |
| **Sami** | 2/14/20 | Initial additions to section 5 | 0.0.01 |
| **Sami** | 2/15/20 | Adds use cases 4.3 and 4.4 | 0.0.02 |
| **Jake** | 2/15/20 | Add use cases 4.1 and 4.2, Sections 1.1 - 1.5 | 0.0.03 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

# Introduction

## Purpose

This document outlines the requirement specifications for the Voting System release version \_\_\_\_. The Voting System is a single stand-alone system. The purpose of this program is to process election ballots and return the results. The program can perform ballot processing for both plurality and single transferable voting (STV) elections.

*<Identify the product whose software requirements are specified in this document, including the revision or release number. Describe the scope of the product that is covered by this SRS, particularly if this SRS describes only part of the system or a single subsystem.>*

## Document Conventions

This document follows the Institute of Electrical and Electronic Engineers (IEEE) software requirements specification (SRS) template. **Bold** text has been used to indicate section headers and to highlight important context.

*<Describe any standards or typographical conventions that were followed when writing this SRS, such as fonts or highlighting that have special significance. For example, state whether priorities for higher-level requirements are assumed to be inherited by detailed requirements, or whether every requirement statement is to have its own priority.>*

## Intended Audience and Reading Suggestions

This project is designed for use by **voting election officials**. In this document, the term **user**, is used to indicate the election official. Other audiences that read this document may include individuals tasked with upholding election integrity, who will read this document to validate the election processing methodology and the required summary information from the ballot processing.

A suggested reading path for the voting election official is to start with section 1. Introduction to get a brief introduction to the product. Secondly, they should continue onto 4. System Features to gain an understanding about the different interactions they may experience with the software and what the expected courses of action are. Lastly, the user should continue onto step 5. Other Non-functional Requirements to gain an understanding about big picture requirements for the program.

A suggested reading path for an election official in charge of validating the program’s design to ensure it upholds the election’s integrity, would be to read the document in a sequential order. The official will gain an introduction to the product, understanding how each functional requirement will work, how the program will work with any other interfaces, as well as how the user will interact with the program and any big picture requirements on the software.

*<Describe the different types of reader that the document is intended for, such as developers, project managers, marketing staff, users, testers, and documentation writers. Describe what the rest of this SRS contains and how it is organized. Suggest a sequence for reading the document, beginning with the overview sections and proceeding through the sections that are most pertinent to each reader type.>*

## Product Scope

The Voting System processes election ballots to determine winner(s) and loser(s) of either a plurality of STV vote. It keeps an audit trail that can be followed to understand any given results of an election. The program returns summary information including the type of vote, the number of ballots, the number of seats, and the number of candidates. It also presents information on the winners and losers accompanied by their respective percentage of votes received (for plurality voting) or the ordering of winners and losers (for STV).

The Voting System is intended to be an easy program to learn how to use, with an expectation that it will take less than 15 minutes to teach a user how to use it.

This product creates many benefits for the election process. To start, the Voting System is reusable for future elections. It is versatile and can be used for both plurality and STV models. The program allows for testing the processing of ballots, as well as automating the election processing task. This automation presumably also decreases the amount of time to process ballots. The tool is also beneficial due to its ability to provide summary information and an audit trail to explain how the results were generated.

*<Provide a short description of the software being specified and its purpose, including relevant benefits, objectives, and goals. Relate the software to corporate goals or business strategies. If a separate vision and scope document is available, refer to it rather than duplicating its contents here.>*

## References

*<List any other documents or Web addresses to which this SRS refers. These may include user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document. Provide enough information so that the reader could access a copy of each reference, including title, author, version number, date, and source or location.>*

# Overall Description

## Product Perspective

*<Describe the context and origin of the product being specified in this SRS. For example, state whether this product is a follow-on member of a product family, a replacement for certain existing systems, or a new, self-contained product. If the SRS defines a component of a larger system, relate the requirements of the larger system to the functionality of this software and identify interfaces between the two. A simple diagram that shows the major components of the overall system, subsystem interconnections, and external interfaces can be helpful.>*

## Product Functions

Main section to handle how the actual voting algorithms will work.

*<Summarize the major functions the product must perform or must let the user perform. Details will be provided in Section 3, so only a high level summary (such as a bullet list) is needed here. Organize the functions to make them understandable to any reader of the SRS. A picture of the major groups of related requirements and how they relate, such as a top level data flow diagram or object class diagram, is often effective.>*

## User Classes and Characteristics

*<Identify the various user classes that you anticipate will use this product. User classes may be differentiated based on frequency of use, subset of product functions used, technical expertise, security or privilege levels, educational level, or experience. Describe the pertinent characteristics of each user class. Certain requirements may pertain only to certain user classes. Distinguish the most important user classes for this product from those who are less important to satisfy.>*

## Operating Environment

*<Describe the environment in which the software will operate, including the hardware platform, operating system and versions, and any other software components or applications with which it must peacefully coexist.>*

## Design and Implementation Constraints

*<Describe any items or issues that will limit the options available to the developers. These might include: corporate or regulatory policies; hardware limitations (timing requirements, memory requirements); interfaces to other applications; specific technologies, tools, and databases to be used; parallel operations; language requirements; communications protocols; security considerations; design conventions or programming standards (for example, if the customer’s organization will be responsible for maintaining the delivered software).>*

## User Documentation

*<List the user documentation components (such as user manuals, on-line help, and tutorials) that will be delivered along with the software. Identify any known user documentation delivery formats or standards.>*

## Assumptions and Dependencies

*<List any assumed factors (as opposed to known facts) that could affect the requirements stated in the SRS. These could include third-party or commercial components that you plan to use, issues around the development or operating environment, or constraints. The project could be affected if these assumptions are incorrect, are not shared, or change. Also identify any dependencies the project has on external factors, such as software components that you intend to reuse from another project, unless they are already documented elsewhere (for example, in the vision and scope document or the project plan).>*

# External Interface Requirements

## User Interfaces

Must have a user interface to collect input parameters to run the program. Can either be text based or GUI based. The interface should be easy to use and trainable to use within 15 minutes.

We can use command line arguments for testing, but these arguments should never be expected from the user. I.e. they should just be able to use the user interface for inputting information.

*<Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear on every screen, keyboard shortcuts, error message display standards, and so on. Define the software components for which a user interface is needed. Details of the user interface design should be documented in a separate user interface specification.>*

## Hardware Interfaces

*<Describe the logical and physical characteristics of each interface between the software product and the hardware components of the system. This may include the supported device types, the nature of the data and control interactions between the software and the hardware, and communication protocols to be used.>*

## Software Interfaces

*<Describe the connections between this product and other specific software components (name and version), including databases, operating systems, tools, libraries, and integrated commercial components. Identify the data items or messages coming into the system and going out and describe the purpose of each. Describe the services needed and the nature of communications. Refer to documents that describe detailed application programming interface protocols. Identify data that will be shared across software components. If the data sharing mechanism must be implemented in a specific way (for example, use of a global data area in a multitasking operating system), specify this as an implementation constraint.>*

## Communications Interfaces

*<Describe the requirements associated with any communications functions required by this product, including e-mail, web browser, network server communications protocols, electronic forms, and so on. Define any pertinent message formatting. Identify any communication standards that will be used, such as FTP or HTTP. Specify any communication security or encryption issues, data transfer rates, and synchronization mechanisms.>*

# System Features

*<This template illustrates organizing the functional requirements for the product by system features, the major services provided by the product. You may prefer to organize this section by use case, mode of operation, user class, object class, functional hierarchy, or combinations of these, whatever makes the most logical sense for your product.>*

*<Don’t really say “System Feature 1.” State the feature name in just a few words.>*

## 4.1 Vote Selection

|  |  |
| --- | --- |
| **Name** | Vote Selection |
| **ID** | user\_01 |
| **Description** | The system should prompt the user to choose the type of voting process the system should perform. The options are plurality or single transferable voting (STV). |
| **Actors** | User |
| **Organizational Benefits** | Allows the system to process multiple methods of conducting an election. This extends the programs usability to be used in both a plurality environment (such as the United States) or a STV environment (such as Ireland). This also allows for the comparison of how an election would result if using one or the other. |
| **Frequency of Use** | User selects vote once per program execution. |
| **Triggers** | User starts the program. |
| **Preconditions** | System must exist. System is running. |
| **Postconditions** | The system understands the correct voting algorithms to use when generating the election results. |
| **Main Course** | 1. System prompts the user for the type of voting process to use    1. 1. Plurality    2. 2. Single Transferable Vote (STV) 2. User enters their choice 3. System records the type of election 4. System moves to use case *Inputting number of seats to fill* (4.2) |
| **Alternate Courses** | AC1 User enters choice of “plurality”   1. System is case insensitive and thus recognizes the selection 2. Return to main course step 3   AC2 User enters choice of “PLURALITY”   1. System is case insensitive and thus recognizes the selection 2. Return to main course step 3   AC3 User enters choice of “2”   1. System recognizes both the text of the selection or the selection number to indicate the choice 2. Return to main course step 3   AC4 User decides they are not ready to run the program anymore   1. User quits program 2. Program terminates |
| **Exceptions** | EX1 User gives invalid input that does not correspond to either of the choices   1. System responds telling the user to please make a selection of either 1. Plurality, or 2. Single Transferable Vote (STV) 2. Return to main course step 1 |

## 4.2 Inputting number of seats to fill

|  |  |
| --- | --- |
| **Name** | Input Number of Seats |
| **ID** | user\_02 |
| **Description** | The system should prompt the user to choose the number of seats to fill in the selection. |
| **Actors** | User |
| **Organizational Benefits** | Allows the system to process election results in which there are a varying number of seats to fill. This promotes the program’s reusability. |
| **Frequency of Use** | The user enters the number of seats once per program execution. |
| **Triggers** | The user has entered a valid vote type from use case 4.1. |
| **Preconditions** | System must exist. System is running. The type of vote has been selected. |
| **Postconditions** | The system understands the number of seats to fill, which determines the number of winners to consider during the election processing. |
| **Main Course** | 1. System prompts the number of seats to fill 2. User enters the number of seats 3. System records the number of seats 4. System moves to use case *Inputting file name(s)* (4.3) |
| **Alternate Courses** | AC1 User decides they are not ready to run the program anymore   1. User opts to quit the program 2. Program terminates |
| **Exceptions** | EX1 User gives a non-numerical input   1. System responds telling the user to please input a positive integer number 2. Return to main course step 1 |

## 4.3 Inputting file name(s)

|  |  |
| --- | --- |
| **Name** | Prompt user to enter file name(s) |
| **ID** | user\_03 |
| **Description** | The user will need to enter the file name(s) that contain the ballots in order for the system to have data to run on. |
| **Actors** | User |
| **Organizational Benefits** | Allows for the system to have the pertinent data to use. The system will be able to correctly tabulate the results and will eliminate the need for manual ballot counting, which is error prone. |
| **Frequency of Use** | Once per election (ELICITATION QUESTION, maybe more?) |
| **Triggers** | System has started. (do we have a flow of what is being asked first so trigger can be more specific? Ex: use case is prompted after algorithm is chosen) |
| **Preconditions** | System has to exist. There must be at least one CSV file in the same directory as the systems program files. |
| **Postconditions** | The system will have the data needed to tabulate a winner. |
| **Main Course** | 1. User is prompted to enter the CSV file name(s) to be ran. 2. User inputs file name(s). 3. System moves onto the next step (what is the next step?) |
| **Alternate Courses** | NONE?? |
| **Exceptions** | EX1 User makes a spelling mistake while inputting one or more files.   1. System informs the user that one or more files were not found. 2. Return to Main Course step 1. |

## 

## 4.4 Shuffle switch

|  |  |
| --- | --- |
| **Name** | Shuffling switch |
| **ID** | STV\_01 |
| **Description** | The system should allow for the option of shuffling the ballots or leaving the ballots unshuffled. When the switch is off (unshuffled), it allows for testing and calibration. When the switch is on (shuffled), it allows for the STV algorithm to work properly. |
| **Actors** | User |
| **Organizational Benefits** | The shuffling switch is used for testing the STV algorithm. This feature will ensure the system is calibrated properly, and allows for unit testing. |
| **Frequency of Use** | Used multiple times between testing and formal execution. |
| **Triggers** | The user has indicated that the ballots should be counted using the STV algorithm. |
| **Preconditions** | System has to exist. The STV algorithm must be chosen by the user. |
| **Postconditions** | The system has either turned on or off the functionality of shuffling ballots. |
| **Main Course** | 1. User is prompted to indicate if shuffling should be turned on or off. 2. User inputs choice. 3. Confirmation is provided to the user that their choice is implemented. |
| **Alternate Courses** | AC1 Plurality algorithm was chosen   1. This use case would be skipped   Q: Is this necessary given the precondition? |
| **Exceptions** | EX1 System fails to interpret user choice   1. System informs the user nothing has changed. 2. Return to Main Course step 1. |

# Other Nonfunctional Requirements--SAMI DO

## Performance Requirements

Election results must be able to be generated in 5 minutes or less, given no more than 100,000 ballots in all files combined.

*<If there are performance requirements for the product under various circumstances, state them here and explain their rationale, to help the developers understand the intent and make suitable design choices. Specify the timing relationships for real time systems. Make such requirements as specific as possible. You may need to state performance requirements for individual functional requirements or features.>*

## ~~Safety Requirements~~

*~~<Specify those requirements that are concerned with possible loss, damage, or harm that could result from the use of the product. Define any safeguards or actions that must be taken, as well as actions that must be prevented. Refer to any external policies or regulations that state safety issues that affect the product’s design or use. Define any safety certifications that must be satisfied.>~~*

## Security Requirements

There are no security issues that the software has to handle. Each individual voting precinct will handle the security of voting itself. The CSV files received have been transmitted via a secure channel.

Q1: What type of secure channel are the files being transmitted by? How do the CSV files end up in the correct directory?

*<Specify any requirements regarding security or privacy issues surrounding use of the product or protection of the data used or created by the product. Define any user identity authentication requirements. Refer to any external policies or regulations containing security issues that affect the product. Define any security or privacy certifications that must be satisfied.>*

## Software Quality Attributes

*<Specify any additional quality characteristics for the product that will be important to either the customers or the developers. Some to consider are: adaptability, availability, correctness, flexibility, interoperability, maintainability, portability, reliability, reusability, robustness, testability, and usability. Write these to be specific, quantitative, and verifiable when possible. At the least, clarify the relative preferences for various attributes, such as ease of use over ease of learning.>*

The software will have interoperability with a Windows CSV file. This allows for streamlined, straight forward usage of the software, where the process of running the software will become more intuitive to the user with future uses.

There will also be functionality to test the STV ballot. This will allow them to ensure the system is calibrated properly and also to allow for unit testing.

## Business Rules

*<List any operating principles about the product, such as which individuals or roles can perform which functions under specific circumstances. These are not functional requirements in themselves, but they may imply certain functional requirements to enforce the rules.>*

**Q1. FOR MONDAY: who is the typical user running the program? Can we assume perfect play on the user end (i.e correctly entering # of seats to fill)?**

**Q2. FOR MONDAY: is there a specific type of person that can run this program?**

# 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**

*<Define all the terms necessary to properly interpret the SRS, including acronyms and abbreviations. You may wish to build a separate glossary that spans multiple projects or the entire organization, and just include terms specific to a single project in each SRS.>*

**Appendix B: Analysis Models**

*<Optionally, include any pertinent analysis models, such as data flow diagrams, class diagrams, state-transition diagrams, or entity-relationship diagrams*.>

**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.>*

-----------------------------------------------------------------------------------------------------------------------------------

**Final Checklist**

* ✅ Name this document SRS\_Team#.xxx where # is your team number and the xxx is the document’s formatting extension (e.g. docx, pdf).
* If you have additional documents, name them using a similar naming convention (e.g. UseCase\_Team1.pdf)
* ✅ Create a subdirectory under your repo-Team# called SRS. This will be where you put ALL documents generated for this assignment. Do not put any files on Canvas.
* ✅ All names with x500 names included on assignment
* Organization:
  + Followed the IEEE Software Requirements Specification Template. Did not add/remove/change sections to document without clear justification.
  + Table of contents is correctly formatted to proper page numbers and titles are followed.
  + Revision history present and filled in.
  + All sections are completed. Consistent font usage.
  + Diagrams, if included, are legible.
* Use Cases:
  + The use cases accurately represent the main functions of the system including the appropriate actors.
  + Use cases are clear and easy to understand.
  + Template is used and followed.
  + Information is not made up.
  + All sections of the use cases are filled in with detail.