

---

# **Software Requirements Specification**

**for**

# **CompuVote**

**Version 1.1 approved**

**Prepared by Aaron Kandikatla, Jack Fornaro, and Nikunj Chawla**

**University of Minnesota – Twin Cities**

**February 2, 2021**

# Table of Contents

|   |            |
|---|------------|
| <b>Table of Contents</b>                      | <b>ii</b>  |
| <b>Revision History</b>                       | <b>iii</b> |
| <b>1. Introduction</b>                        | <b>1</b>   |
| 1.1 Purpose                                   | 1          |
| 1.2 Document Conventions                      | 1          |
| 1.3 Intended Audience and Reading Suggestions | 1          |
| 1.4 Product Scope                             | 4          |
| 1.5 References                                | 4          |
| <b>2. Overall Description</b>                 | <b>6</b>   |
| 2.1 Product Perspective                       | 6          |
| 2.2 Product Functions                         | 7          |
| 2.3 User Classes and Characteristics          | 7          |
| 2.4 Operating Environment                     | 7          |
| 2.5 Design and Implementation Constraints     | 12         |
| 2.6 User Documentation                        | 13         |
| 2.7 Assumptions and Dependencies              | 13         |
| <b>3. External Interface Requirements</b>     | <b>14</b>  |
| 3.1 User Interfaces                           | 14         |
| 3.2 Hardware Interfaces                       | 15         |
| 3.3 Software Interfaces                       | 16         |
| 3.4 Communications Interfaces                 | 19         |
| <b>4. System Features</b>                     | <b>19</b>  |
| 4.1 Input File Location                       | 20         |
| 4.2 Read and Parse File                       | 21         |
| 4.3 Run IR Election                           | 27         |
| 4.4 Run OPL Election                          | 28         |
| 4.5 Produce Audit File                        | 30         |
| 4.6 Produce Report File                       | 32         |
| 4.7 Display Summary to Screen                 | 34         |
| <b>5. Other Nonfunctional Requirements</b>    | <b>35</b>  |
| 5.1 Performance Requirements                  | 35         |
| 5.2 Safety Requirements                       | 36         |
| 5.3 Security Requirements                     | 36         |
| 5.4 Software Quality Attributes               | 36         |
| 5.5 Business Rules                            | 37         |
| <b>6. Other Requirements</b>                  | <b>39</b>  |
| <b>Appendix A: Glossary</b>                   | <b>39</b>  |
| <b>Appendix B: Analysis Models</b>            | <b>43</b>  |

## Revision History

| <b>Name</b>             | <b>Date</b> | <b>Reason For Changes</b>              | <b>Version</b> |
|-------------------------|-------------|--|----------------|
| Nikunj, Aaron, and Jack | 2/16        | Initial version of the complete SRS    | 1.0            |
| Nikunj, Aaron, and Jack | 2/19        | Minor changes to the SRS and use cases | 1.1            |

# 1. Introduction

## 1.1 Purpose

*<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.>*

The purpose of this document is to present an overview of the CompuVote Election System Version 1.0. The system presented can be used by anyone who needs to hold an **Instant Runoff Election (IR)** or an **Open Party Listing (OPL)** election. The CompuVote System is rather standalone, but needs a **Comma-Separated-Values file (CSV file)** that includes ballot and election **input** information. The operations in the system will parse the **CSV file**, perform all the calculations and results needed and **output** those to a formatted report file, an **audit file**. A summary of the election will be printed to the screen after every run.

## 1.2 Document Conventions

- **Bold** is used for glossary terms.
- The `Courier New` font is used for code and **output** and can be used for **file paths**, **directories**, and **file names**.

## 1.3 Intended Audience and Reading Suggestions

*<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.>*

- The document begins with introducing the system, the scope, and the references used. It then goes on to describe the product, its requirements for use, implementation and constraints, and documentation. After which, the document explains the ways in which CompuVote **interfaces** with the user, **OS**, and **hardware**. The document then goes into detail about the functional features of the system and the nonfunctional features and constraints. Finally, at the end of the document, there are appendices available: a glossary, any analysis models used, and a list of to-be-determined references.
- Note: For recommended reading order, section 1 subsections 1.1 – 1.4 are applicable to all readers. They should be read to understand the higher-level details regarding CompuVote. These sections should be read before each of the below reading orders per reader.
- Developers who are looking to maintain or extend the **software**
  - Recommended Reading Order:
    - Section 6
      - Appendix A

- Section 2
  - Subsection 2.1
  - Subsection 2.2
  - Subsection 2.3
  - Subsection 2.4
  - Subsection 2.5
  - Subsection 2.6
  - Subsection 2.7
- Section 6
  - Appendix B
- Section 3
  - Subsection 3.1
  - Subsection 3.2
  - Subsection 3.3
- Section 4 (as needed)
- Section 5
  - Subsection 5.1
  - Subsection 5.2
  - Subsection 5.3
  - Subsection 5.4
  - Subsection 5.5
- Testers who are responsible for ensuring that CompuVote functions as expected
  - Recommended Reading Order:
    - Section 6
      - Appendix A
    - Section 2
      - Subsection 2.1
      - Subsection 2.2
      - Subsection 2.3
      - Subsection 2.4
      - Subsection 2.5
      - Subsection 2.6
      - Subsection 2.7
    - Section 6
      - Appendix B
    - Section 3
      - Subsection 3.1
      - Subsection 3.3
    - Section 4 (as needed)
    - Section 5
      - Subsection 5.1
      - Subsection 5.5
- Documentation writers who create documentation for additions or extensions of the provided **software**

- Recommended Reading Order:
  - Section 6
    - Appendix A
  - Section 2
    - Subsection 2.1
    - Subsection 2.2
    - Subsection 2.3
    - Subsection 2.4
    - Subsection 2.6
    - Subsection 2.7
  - Section 6
    - Appendix B
  - Section 3
    - Subsection 3.1
    - Subsection 3.2
    - Subsection 3.3
  - Section 4 (as needed)
  - Section 5
    - Subsection 5.1
    - Subsection 5.2
    - Subsection 5.3
    - Subsection 5.5
- Election officials who are responsible for tallying ballots for an election involving one of the instant runoff voting or open party list voting systems and showing how the process occurred
  - Recommended Reading Order:
    - Section 6
      - Appendix A
    - Section 2
      - Subsection 2.1
      - Subsection 2.2
      - Subsection 2.3
      - Subsection 2.4
      - Subsection 2.6
      - Subsection 2.7
    - Section 6
      - Appendix B
    - Section 3
      - Subsection 3.1
      - Subsection 3.3
    - Section 4 (as needed)
    - Section 5
      - Subsection 5.1
      - Subsection 5.2

- Subsection 5.3
  - Subsection 5.5
- Media personnel who will be reporting on the results of the election
  - Recommended Reading Order:
    - Section 6
      - Appendix A
    - Section 2
      - Subsection 2.1
      - Subsection 2.2
      - Subsection 2.3
    - Section 6
      - Appendix B
    - Section 3
      - Subsection 3.1 (as needed)

## 1.4 Product Scope

*<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.>*

CompuVote is a Java program that counts votes to get election results for two voting systems: **Instant Runoff Voting** and **Open Party List Voting**. The purpose of the product is to provide fast and reliable **algorithms** that can be used by various election systems and includes the infrastructure to easily support the addition of other voting systems. For more information about the qualities of the product, see section 5.4 (Software Quality Attributes). The system can be used for any scenario that involves elections and tallying ballots in the specified format. Businesses or corporations can also use CompuVote for their own internal voting systems. CompuVote can also generate a report file containing a summary of an election, which can be sent to the media by an election official.

## 1.5 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.>*

1.1. What Is Ubuntu?, [help.ubuntu.com/lts/installation-guide/s390x/ch01s01.html](http://help.ubuntu.com/lts/installation-guide/s390x/ch01s01.html).

“AppsToGo: Use UMN Apps on Your Personal Device.” *AppsToGo: Use UMN Apps on Your Personal Device | IT@UMN | The People behind the Technology*, [it.umn.edu/services-technologies/self-help-guides/appstogo-use-umn-apps-your-personal](http://it.umn.edu/services-technologies/self-help-guides/appstogo-use-umn-apps-your-personal).

Chadwick, Ryan. "Linux Tutorial." *A Collection of Technology Tutorials*, ryanstutorials.net/linuxtutorial/.

Chawla, Nikunj. "Formatting." *GitHub*, github.umn.edu/chawl025/nikunjJavaFormattingFiles/wiki.

Chawla, Nikunj. "Repo-Team19-Doc." *Github.io*, nik312123.github.io/repo-Team19-doc/.

"Difference between 32-Bit and 64-Bit Operating Systems." *GeeksforGeeks*, 7 Oct. 2020, www.geeksforgeeks.org/difference-32-bit-64-bit-operating-systems/.

"Duo Two Factor Authentication." *Duo Two Factor Authentication | IT@UMN | The People behind the Technology*, it.umn.edu/services-technologies/duo-two-factor-authentication.

Eross-Msft. "PowerShell." *Microsoft Docs*, docs.microsoft.com/en-us/windows-server/administration/windows-commands/powershell.

Fornaro, Chawla, Kandikatla. "Repo-Team19 GitHub." *GitHub*, github.umn.edu/umn-csci-5801-S21-002/repo-Team19.

Fornaro, Chawla, Kandikatla. "Use Cases." *Google Drive*, docs.google.com/document/d/14JIVOzj5JTdkJXPeuccESReYUo8uLF8LDRVpnKvbAnA/edit?usp=sharing.

Gee, Trisha. "Writing Tests with JUnit 5 – IntelliJ IDEA Blog: JetBrains." *JetBrains Blog*, 12 Feb. 2021, blog.jetbrains.com/idea/2020/09/writing-tests-with-junit-5/.

IEEE Software Requirements Specification Template - Karl E. Weigers 1999, https://web.cs.dal.ca/~hawkey/3130/srs\_template-ieee.doc

"IRV Audit Example." *Pastebin*, pastebin.com/za0DpV8K.

"IRV Report Example." *Pastebin*, pastebin.com/StX3Et9P.

Kauffman, Chris. *Accessing Unix/Linux Programming Environments*, www-users.cs.umn.edu/~kauffman/tutorials/unix-environment.html.

Naushad, Alok. "Basic Linux Commands for Beginners: Linux." *Maker Pro*, Maker Pro, 17 Feb. 2021, maker.pro/linux/tutorial/basic-linux-commands-for-beginners.

"OPL Audit Example." *Pastebin*, pastebin.com/eWMHuTvK.

"OPL Report Example." *Pastebin*, pastebin.com/19rdgmgx.

Stefan Bechtold, Sam Brannen. *JUnit 5 User Guide*, junit.org/junit5/docs/current/user-guide/.

"TechTerms." *The Tech Terms Computer Dictionary*, techterms.com/.

"The Linux Command Line for Beginners." *Ubuntu*, ubuntu.com/tutorials/command-line-for-beginners.



“Use Secure Shell (SSH).” *Use Secure Shell (SSH) | IT@UMN | The People behind the Technology*, [it.umn.edu/services-technologies/resources/use-secure-shell-ssh](http://it.umn.edu/services-technologies/resources/use-secure-shell-ssh).

“Virtual Online Linux Environment Self-Help Guide.” *Virtual Online Linux Environment Self-Help Guide | Information Technology | College of Science and Engineering*, [cse.umn.edu/cseit/virtual-online-linux-environment-self-help-guide](http://cse.umn.edu/cseit/virtual-online-linux-environment-self-help-guide).

*What Is a Path?*, 3 Sept. 2019, [www.computerhope.com/jargon/p/path.html](http://www.computerhope.com/jargon/p/path.html).

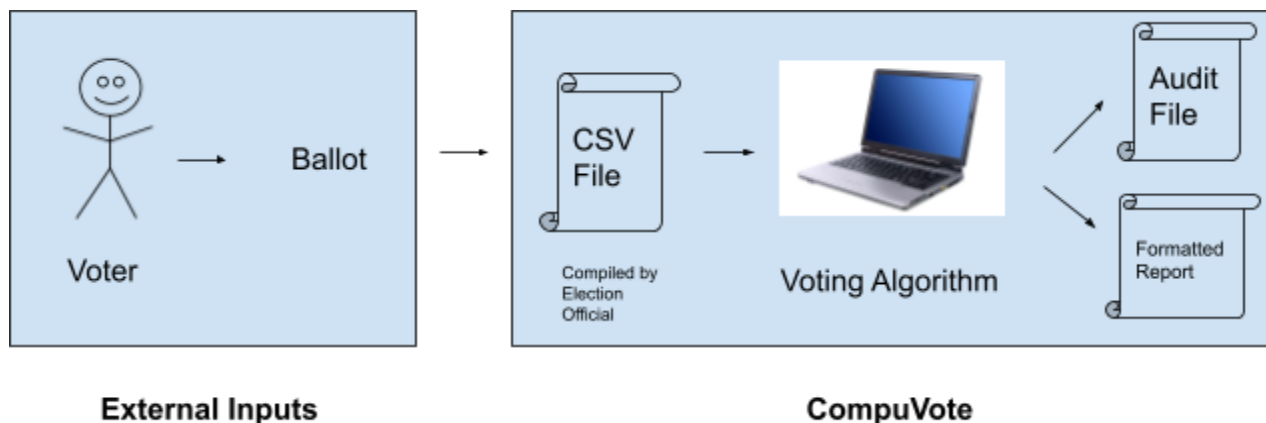
“What Is the Current Directory?” *What Is the Current Directory? -- Definition by The Linux Information Project (LINFO)*, Linux Information Project (LINFO), [www.linfo.org/current\\_directory.html](http://www.linfo.org/current_directory.html).

## 2. Overall Description

### 2.1 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.>

The CompuVote system is a replacement for inefficient voting systems. Voters will cast their ballot, and each ballot will be collected and compiled into a **CSV file** by the election officials. This file will then be read in by the **algorithm**, and once the election is complete, **output** files will be produced. The **output** will be in the form of an **audit file**, displaying a rough recreation of the election and enough information so the election could be run again. There will also be a formal report, readable by anyone, that will be sent out to the media for analysis, and a summary of the election will be printed to the screen.



## 2.2 Product Functions

*<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.>*

There is not much the user can do to interact with the system. The user, in most cases, is the election official. They will **input** the **CSV file**, and then the system will run an election. The user can then use the information (**audit file** for programmers and the election official, and a formatted report for anyone, as well as a summary of the election printed to the screen) for further distribution and analysis.

## 2.3 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.>*

- User class level 1 (Access to source code):
  - Developers involved in maintaining and/or extending the **software**
  - Testers who are in charge in assuring that the **software** functions properly
  - Documentation writers who create documentation for additions or extensions of the provided **software**
- User class level 2 (Access to run the program):
  - Election officials who have access to officially execute CompuVote and authority to share reports exported by the **software**
- User class level 3 (Access to the report file):
  - Media personnel who have access to the official election reports (produced by CompuVote) given by an election official

## 2.4 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.>*

- Minimum recommendations:
  - 16+ GB **RAM**
  - **Windows** 10 1909 or later, **Ubuntu** 16.04 LTS or later, or **macOS** 10.13.6 or later
    - Of these, **Ubuntu** 18.04 LTS or later is recommended
  - **32-bit** and **64-bit** systems both supported
  - **Java JDK** 11.0.9+

- Intel Core i5 or better/more recent Intel-based, 3+ processors, 800+ Hz each
- Dell system recommended
- Machines Available
  - Location: Civil Engineering 230
    - Machine names: csel-ce230-01.cselabs.umn.edu to csel-ce230-43.cselabs.umn.edu
      - **Windows** 10
      - Dell Precision 3630
      - Intel® Core™ i7 @ 3.2GHz (x6)
      - 32 GB **RAM**
  - Location: Frontier Hall 126
    - Machine names: csel-fh126-01.cselabs.umn.edu to csel-fh126-06.cselabs.umn.edu
      - **Windows** 10 (3), **Ubuntu** 18.04 (1)
      - Dell Precision T3420
      - Intel® Core™ i5 @ 3.4GHz (x4)
      - 32 GB **RAM**
  - Location: Keller Hall 1-200
    - Machine names: csel-kh1200-01.cselabs.umn.edu to csel-kh1200-29.cselabs.umn.edu
      - **Windows** 10
      - Dell OptiPlex 9020
      - Intel® Core™ i7 @ 4.2GHz (x4)
      - 32 GB **RAM**
  - Location: Keller Hall 1-202
    - Machine names: csel-robovis1.cselabs.umn.edu, csel-robovis2.cselabs.umn.edu
      - **Ubuntu** 16.04
      - Dell XPS 8910
      - Intel Core i7-6700 CPU 3.40GHz
      - 16GB **RAM**
      - Nvidia GeForce GTX 1080
  - Location: Keller Hall 1-250
    - Machine names: csel-kh1250-01.cselabs.umn.edu to csel-kh1250-37.cselabs.umn.edu
      - **Ubuntu** 20.04
      - Dell Optiplex 9020
      - Intel® Core™ i7 @ 900Hz (x3)
      - 32 GB **RAM**
  - Location: Keller Hall 1-254
    - Machine names: csel-kh1254-01.cselabs.umn.edu to csel-kh1254-19.cselabs.umn.edu
      - **Windows** 10
      - Dell Precision T7910

- Xeon @ 3GHz (x12)
- 32 GB **RAM**
- Location: Keller Hall 1-260
  - Machine names: csel-kh1260-01.cselabs.umn.edu to csel-kh1260-20.cselabs.umn.edu
    - **Ubuntu** 20.04
    - Dell Precision 3630
    - Intel Core i7-4790 (Quad-Core) 3.60GHz
    - 32 GB **RAM**
    - DVD+/-RW
    - Intel® UHD Graphics 630
- Location: Keller Hall 1-262
  - Machine names: csel-kh1262-01.cselabs.umn.edu to csel-kh1262-28.cselabs.umn.edu
    - **Ubuntu** 20.04
    - Dell OptiPlex 7050
    - Intel Core i7-7700 3.6GHz
    - 32 GB **RAM**
    - DVD+/-RW
    - Intel® UHD Graphics 630
- Location: Keller Hall 2-216 - Graduate Lab
  - Machine names: coachz.cs.umn.edu, strongbad.cs.umn.edu
    - **macOS** 10.13.6
    - Intel Core i5 3.2 GHz
    - 16 GB **RAM**
    - DVD-RW
    - 2 x NVIDIA® GeForce GT 755M 1024MB
  - Machine names: exa.cs.umn.edu, femto.cs.umn.edu, galaxy.cs.umn.edu, nebula.cs.umn.edu, pulsar.cs.umn.edu, quasar.cs.umn.edu
    - **Ubuntu** 16.04
    - Intel Core i7-4770 3.40GHz
    - 16 GB **RAM**
    - DVD-RW
    - AMD Radeon HD 8570
  - Machine names: cs-cello.cs.umn.edu, cs-flute.cs.umn.edu, cs-oboe.cs.umn.edu, cs-saxophone.cs.umn.edu, cs-trombone.cs.umn.edu, cs-trumpet.cs.umn.edu, cs-violin.cs.umn.edu
    - **Ubuntu** 16.04
    - Intel Xeon e5-2460 2.4ghz 20 cores
    - 32 GB **RAM**
    - DVD-RWX drive
    - Nvidia Quadro M2000

- Machine names: cs-atto.cs.umn.edu, cs-carbon.cs.umn.edu, cs-deca.cs.umn.edu, cs-deci.cs.umn.edu, cs-hecto.cs.umn.edu, cs-myria.cs.umn.edu, cs-neon.cs.umn.edu
  - **Windows 10**
  - Intel Xeon e5-2460 2.4ghz 20 cores
  - 32 GB **RAM**
  - DVD-RWX drive
  - Nvidia Quadro M2000
- Location: Keller Hall 4-240
  - Machine names: csel-kh4240-01.cselabs.umn.edu to csel-kh4240-10.cselabs.umn.edu
    - **Ubuntu 20.04**
    - Dell OptiPlex 9020
    - Intel® Core™ i7 @ 1.6GHz (x3)
    - 32 GB **RAM**
- Location: Keller Hall 4-250
  - Machine names: csel-kh4250-01.cselabs.umn.edu to csel-kh4250-49.cselabs.umn.edu
    - **Ubuntu 20.04**
    - Dell Optiplex 9020
    - Intel® Core™ i7 @ 1.6 GHz (x3)
    - 32 GB **RAM**
- Location: Lind Hall 150 - Taylor Center
  - Machine names: csel-lind150-01.cselabs.umn.edu to csel-lind150-50.cselabs.umn.edu
    - **Windows 10**
    - Dell Precision T3420
    - Intel® Core™ i5 @ 3.4GHz (x4)
    - 64 GB **RAM**
- Location: Lind Hall 40
  - Machine names: csel-lind40-01.cselabs.umn.edu to csel-lind40-43.cselabs.umn.edu
    - **Ubuntu 20.04**
    - Dell Precision T3620
    - Intel® Core™ i5 @ 800Hz (x3)
    - 32 GB **RAM**
- Location: Anderson Design Lab ME 2134
  - Machine names: csel-me2134-01.cselabs.umn.edu, csel-me2134-02.cselabs.umn.edu, csel-me2134-03.cselabs.umn.edu, csel-me2134-04.cselabs.umn.edu
    - **Windows 10**
    - Dell OptiPlex 9020
    - Intel Core (Quad-core) i7 @ 3.6 GHz
- Location: Mechanical Engineering 302

- Machine names: csel-me302-01.cselabs.umn.edu to csel-me302-18.cselabs.umn.edu
  - **Windows** 10
  - Dell Precision 3630
  - Intel® Core™ i7 @ 3.2GHz (x6)
  - 32 GB **RAM**
- Location: Mechanical Engineering 308
  - Machine names: csel-me308-01.cselabs.umn.edu to csel-me308-30.cselabs.umn.edu
    - **Windows** 10
    - Dell Precision T3420
    - Intel® Core™ i5 @ 3.4GHz (x4)
    - 64 GB **RAM**
- Location: Mechanical Engineering 314
  - Machine names: csel-me314-01.cselabs.umn.edu to csel-me314-35.cselabs.umn.edu
    - **Windows** 10
    - Dell Precision Tower 3620
    - Intel® Core™ i5 @ 4.2GHz (x4)
- Location: Remote only
  - Machine names: cuda01.cselabs.umn.edu to cuda08.cselabs.umn.edu
    - **Ubuntu** 16.04
    - Dell Precision T5400
    - 2 x Quad-Core Intel Xeon 2.00GHz
    - 8 GB **RAM**
    - NVIDIA GeForce graphics for CUDA programming
  - Machine names: phi01.cselabs.umn.edu to phi08.cselabs.umn.edu
    - **Ubuntu** 20.04
    - Dell Power Edge R720
    - 2 x 8-Core Intel Xeon 2.00GHz
    - 64 GB **RAM**
    - Xeon Phi cards for MIC Architecture programming
  - Machine names: dio.cs.umn.edu
    - **Ubuntu** 16.04
    - Dell PowerEdge 6950
    - 4 x Dual-Core AMD Opteron Processor 8216 2.40GHz
    - 16 GB **RAM**
    - 1 TB Hard Drive
  - Machine names: maximus.cs.umn.edu
    - **Ubuntu** 16.04
    - Dell PowerEdge R815
    - 4 x AMD® Opteron® 6220 (Eight-Core) @ 3.00GHz
    - 192 GB **RAM**
    - 1 TB Hard Drive

- Location: Walter Library 103
  - Machine names: csel-w103-01.cselabs.umn.edu to csel-w103-29.cselabs.umn.edu
    - **Windows 10**
    - Dell Precision 3630
    - Intel® Core™ i7 @ 3.2GHz (x6)
    - 32 GB **RAM**
  - Machine names: csel-w103-30.cselabs.umn.edu to csel-w130-58.cselabs.umn.edu
    - **Ubuntu 20.04**
    - Dell Precision T1700
    - Intel® Core™ i7 @ 3.6GHz (x4)
    - 32 GB **RAM**
- Location: Walter Library 106
  - Machine names: csel-w106-01.cselabs.umn.edu to csel-w106-33.cselabs.umn.edu
    - **Ubuntu 20.04**
    - Dell Optiplex 9020
    - Intel® Core™ i7 @ 3.5GHz (x3)
    - 32 GB **RAM**

## 2.5 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).>*

- See 2.4 for program running requirements.
- See 2.7 for constraints regarding Java version for development.
- Also, see 2.7 for constraints regarding **unit tests** developed for the project (using **JUnit**). This constraint only applies to **unit tests** developed for CompuVote as delivered by us. If you wish to use another **unit testing** framework/library, then you would be unable to use our **unit tests** in the maintenance or extension of CompuVote.
- See 3.3 for applications used to interact with CompuVote and how to use them.
- For the development of the project, the following formatting standards were used and may be good to adhere to for consistency:  
<https://github.umn.edu/chaw1025/nikunjJavaFormattingFiles/wiki>.
- Additionally, as mentioned in the aforementioned formatting standards document, the majority of formatting standards not covered on the page are stored in profiles made for use with **IntelliJ IDEA**. As such, if the use of those profiles is desired, then the **IntelliJ IDEA IDE** must be used.

## 2.6 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.>*

- This document provides the formatting standards used for further development of CompuVote (e.g. maintenance and extension):  
<https://github.umn.edu/chaw1025/nikunjJavaFormattingFiles/wiki>
- This address will likely hold the documentation for each class, method, and class field in the classes we use for our project: See 6.3
- This is the **repository** at which our code will be stored, which also includes a **README file** that will specify how to use CompuVote:  
<https://github.umn.edu/umn-csci-5801-S21-002/repo-Team19>
- This is the user guide to **JUnit**, which we used for developing **unit tests** as well as an **IntelliJ**-specific document used to explain how to use **JUnit** properly in **IntelliJ**:  
<https://junit.org/junit5/docs/current/user-guide/>,  
<https://blog.jetbrains.com/idea/2020/09/writing-tests-with-junit-5/>
- Here are three guides on how to use the **command line**. Many items in each of the documents are not required for use of CompuVote but are still useful to know about:  
<https://ubuntu.com/tutorials/command-line-for-beginners>,  
<https://ryantutorials.net/linuxtutorial/>,  
<https://maker.pro/linux/tutorial/basic-linux-commands-for-beginners>.

## 2.7 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).>*

- Java 11.0.9 was used for the development of the project. As such, Java 11.0.9 or higher is required for running the program. However, **JDK 11.0.9** is specifically required for development of the project if the Java version for usage of Java 11.0.9 is to be kept.
- **JUnit 5.7.0** and its dependencies were used for developing **unit tests** in the project. (Particularly, we installed **JUnit** from Maven with the name org.junit.jupiter:junit-jupiter:5.7.0 and downloaded sources, JavaDocs, annotations, and transitive dependencies.)
- **IntelliJ 2020.3.2** was used in the development of the project along with the corresponding CheckStyle-IDEA plugin of version 5.47.0.
- See 2.4 for operating environment constraints.



## 3. External Interface Requirements

### 3.1 User Interfaces

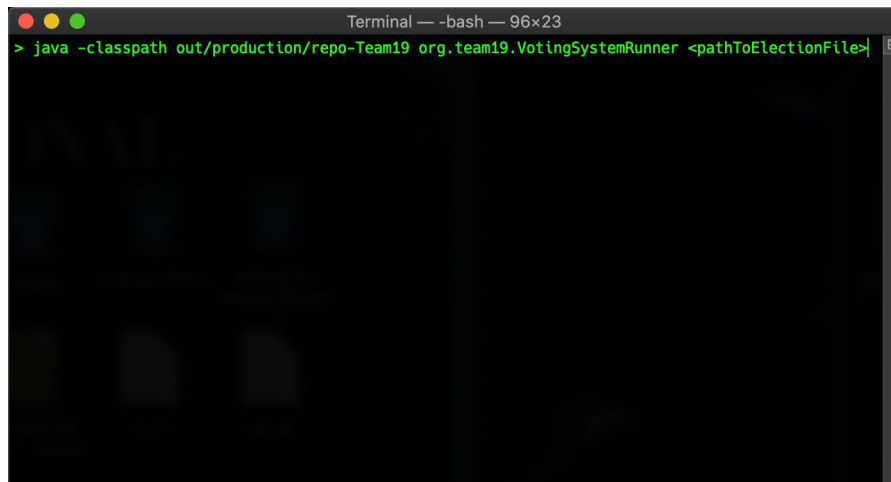
*<Describe the logical characteristics of each interface between the software product and the users. This may include sample screen images, any GUI standards or product family style guides that are to be followed, screen layout constraints, standard buttons and functions (e.g., help) that will appear 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.>*

As long as your system meets requirements specified in 2.4, there should be no additional screen layout constraints.

In terms of error message display standards, if an error occurs in the running of the program, then the program will print a message in the format “Error: <message>” without quotes, where <message> is replaced with a detailed error message describing why the error occurred. This message will be printed to standard error. Then, the program will exit with a nonzero status.

The **user interface** will be run through the appropriate **CLI** for your **OS**. See 3.3 for more details on an example of an **OS**-specific **CLI** that can run CompuVote.

Assuming you have Java 11.0.9 or higher as described in the requirements, you can expect to run something very similar to the below command in the downloaded `repo-Team19` **directory**:

A screenshot of a terminal window with a dark background. The title bar at the top reads "Terminal — -bash — 96x23". The command prompt shows a green prompt character followed by the command: `java -classpath out/production/repo-Team19 org.team19.VotingSystemRunner <pathToElectionFile>`. The cursor is at the end of the command line.

```
Terminal — -bash — 96x23
> java -classpath out/production/repo-Team19 org.team19.VotingSystemRunner <pathToElectionFile>
```

<pathToElectionFile> would be replaced with the **file path** to the election file on which to run CompuVote. This can be an **absolute path** (that is, relative to `C:\` or another **disk drive** on **Windows** or `/` on **macOS** and **Ubuntu**) or a path relative to the `repo-Team19` **directory**. (Refer to <https://www.computerhope.com/jargon/p/path.htm> for more information on what **file paths** are and how they work.)

Then, when this command is entered, it will tally the ballots and display a summary of the election and statistics to the **CLI** along with other tasks such as the audit and report file, which are unrelated to the **UI**.

## 3.2 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.>*

There are several kinds of **hardware** interfaces that can be used to interact with CompuVote. CompuVote is made to interface with the CSE Lab Machines mentioned in detail in 2.4. There are several methods to access these CSE Lab Machines:

- You can access the machines by physically travelling to the locations mentioned in 2.4 on the University of Minnesota – Twin Cities campus, logging onto a workstation with a UMN internet ID and password, downloading the repo-Team19 zip/**directory**, and following what is mentioned in 3.1.
- You can access **Ubuntu** machines by use of **VOLE**. **VOLE** delivers the capabilities of a CSE desktop virtually online with a graphical **user interface** for the **OS**. The details regarding how to use **VOLE** and the access to **VOLE** can be found here: <https://cse.umn.edu/cseit/virtual-online-linux-environment-self-help-guide>
  - Note that **Duo** is required to use **VOLE**. Details on **Duo** can be found here: <https://it.umn.edu/services-technologies/duo-two-factor-authentication>.
- You can access **Ubuntu** machines and **macOS** machines by the use of **SSH** with the corresponding machine names provided in 2.4. You can learn more about **software** that can be used to use **SSH** here: <https://it.umn.edu/services-technologies/resources/use-secure-shell-ssh>. For the use of **SSH** on **Ubuntu** and **macOS**, you may also refer to <https://www-users.cs.umn.edu/~kauffman/tutorials/unix-environment.html>. For the use of **SSH** on **Windows**, you may refer to <https://www.ssh.com/ssh/putty/windows/>.
  - Note that **Duo** is required to use **SSH**. Details on **Duo** can be found here: <https://it.umn.edu/services-technologies/duo-two-factor-authentication>.
- You can access **Windows** machines by the use of **Citrix Receiver** (Workspace) and **Kumo**. To learn more about how to do that, please refer to <https://it.umn.edu/services-technologies/self-help-guides/appstogo-use-umn-apps-your-personal>.
  - Note that **Duo** is required to use UMN apps. Details on **Duo** can be found here: <https://it.umn.edu/services-technologies/duo-two-factor-authentication>.

Other than that, CompuVote interfaces with the **hardware** through File **I/O**. It reads (or attempts to read) from the file provided via **command line arguments**. It **outputs** summary information to the **CLI** (standard **output**), to an **audit file** in the `audits` **directory** (which is directly under the `repo-Team19` **directory**), and to a report file in the `reports` **directory** (which is also directly under the `repo-Team19` **directory**).

### 3.3 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.>*

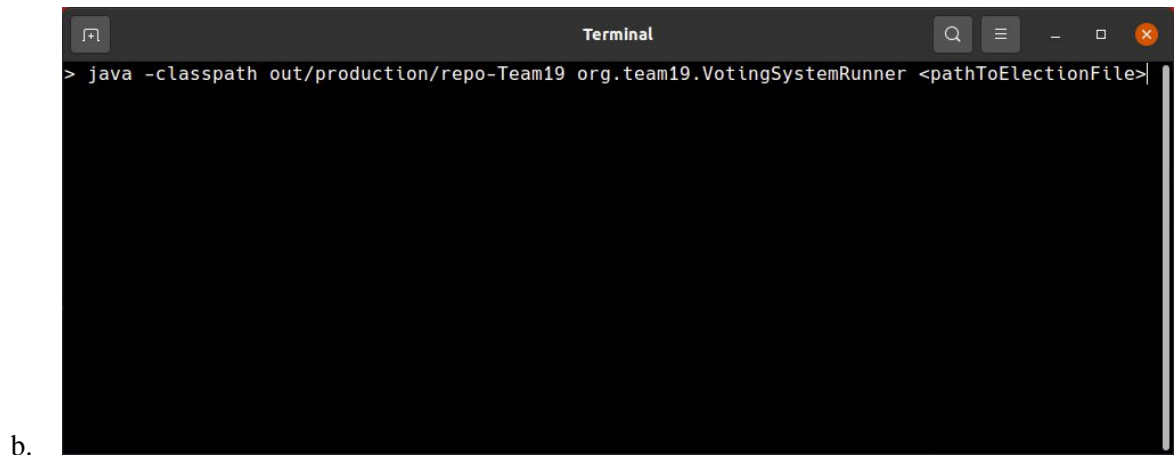
The **software** interface used to interact with CompuVote varies depending on the **OS**, though the general idea behind the command is the same on each **OS** as long as the requirements specified in 2.4 are met.

In terms of **software** interface, **CLI** is used. There are no graphical components to the **user interface**. You enter a variation of the command shown in the image shown in 3.1. The below steps assume you have worked through the **hardware** interfaces required to login to the system (see 3.2).

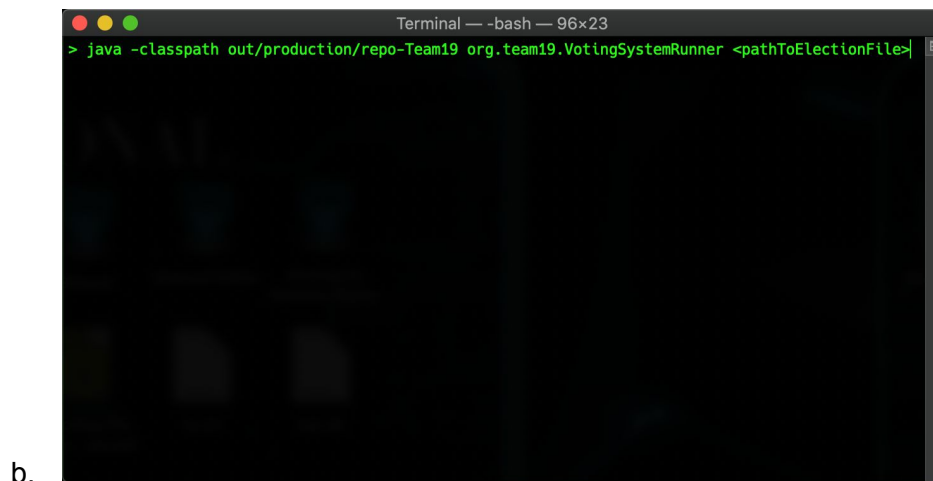
Let P be a step that is required when working directly/physically with the machine, V be a step that is required when using **VOLE**, S be a step that is required when using **SSH**, and A be a step that is required when using UMN Apps. The lack of one of P, V, S, or A for an **OS** indicates the interface corresponding to the missing letter cannot be used.

#### Ubuntu:

- One method of interacting with CompuVote on **Ubuntu** is by usage of the gnome-terminal application.
- Steps:
  1. (P) Click the nine squares in a 3-by-3 grid in the lower-left
  2. (P) Search **Terminal**, and open the application labeled **Terminal**. (gnome-terminal 3.18.3 or higher is required.)
  3. (V) Open the **Terminal** application at the bottom (the black application with the dollar sign and underscore).
  4. (V) Click File → New Tab
  5. (P, V, S) Use the `cd` command to navigate to the `repo-Team19` **directory**
    - a. (P, V, S) Run `java -classpath out/production/repo-Team19 org.team19.VotingSystemRunner <pathToElectionFile>` where `<pathToElectionFile>` would be replaced with the **file path** to the election file on which to run CompuVote. This can be an **absolute path** (that is, relative to / on **Ubuntu**) or a path relative to the `repo-Team19` **directory**.

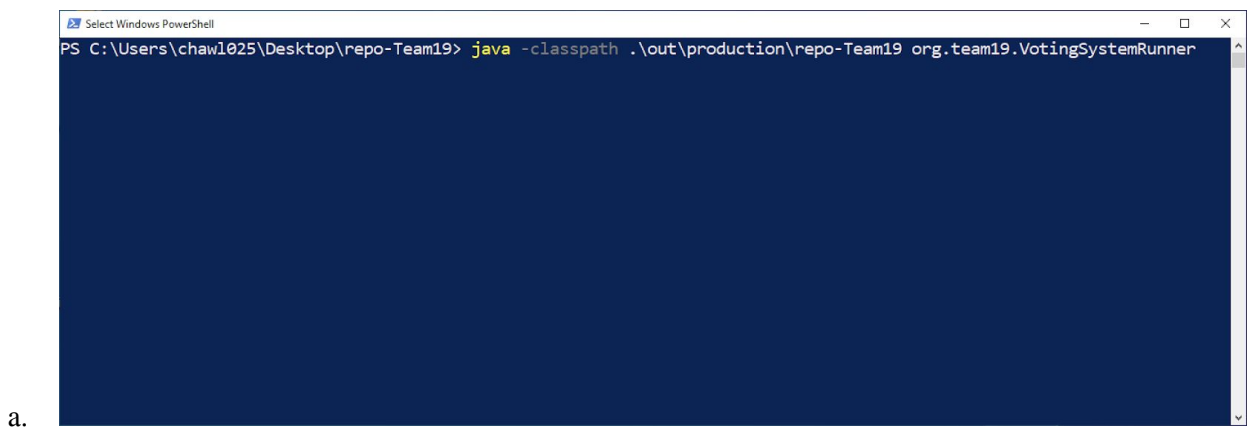
**macOS:**

- One method of interacting with CompuVote on **macOS** is by usage of the **Terminal** application.
- Steps:
  1. (P) Navigate to `/Applications/Utilities`
  2. (P) Open `Terminal.app` (2.8.3 or higher required)
  3. (P, S) Use the `cd` command to navigate to the **repo-Team19 directory**
    - a. (P, S) Run `java -classpath out/production/repo-Team19 org.team19.VotingSystemRunner <pathToElectionFile>` where `<pathToElectionFile>` would be replaced with the **file path** to the election file on which to run CompuVote. This can be an **absolute path** (that is, relative to `/` on **macOS**) or a path relative to the **repo-Team19 directory**.

**Windows:**

- One method of interacting with CompuVote on **Windows** is by usage of the **Powershell** application.

1. (P, A) Press the **Windows** key so the search comes up, and type in “**Powershell**” (without the quotes)
2. (P, A) Choose the application **Windows Powershell** (NOT to be confused with **Windows Powershell** (x86)). (Note: **Powershell** 5.1+ is required.)
3. (P, A) Use the **Powershell** command `Set-Location` (very similar to `cd` on **Ubuntu** and **macOS**) to navigate to the `repo-Team19` **directory**
4. (P, A) Run `java -classpath .\out\production\repo-Team19 org.team19.VotingSystemRunner <pathToElectionFile>` where `<pathToElectionFile>` would be replaced with the **file path** to the election file on which to run CompuVote. This can be an **absolute path** (that is, relative to `C:\` on **Windows**) or a path relative to the `repo-Team19` **directory**.



The data coming into the CompuVote system is the filepath of the election file. This can be an **absolute path** (that is, relative to `C:\` or another **disk drive** on **Windows** or `/` on **macOS** and **Ubuntu**) or a path relative to the `repo-Team19` **directory**. (Refer to <https://www.computerhope.com/jargon/p/path.htm> for more information on what **file paths** are and how they work.)

As far as **output** of the system, there are two possibilities:

- An error message if something problematic occurs in the command-line **arguments**, retrieving the election file, parsing the election file, or writing the audit or report files.
- The audit and report files and the summary **output**:
  - **Audit file**: Gives an in-detail explanation of essentially everything that occurred in the election, including but not limited to the tallying of every ballot, the type of voting, number of candidates, candidates themselves and their parties, number of ballots, number of seats (where applicable), any calculations that occur for the election, how many votes each candidate had, the percentage of votes each candidate had, and any information pertaining to the running of the election.
  - Report file: Gives a summary of what occurred in the election, including but not limited to the type of election, number of candidates, candidates themselves and

their parties, number of ballots, number of seats (where applicable), how many votes each candidate had, and the percentage of votes each candidate had. This is oriented towards a reporting source such as the media.

- Summary **output**: Gives a summary of the results of the election with the same information as the report to the **CLI** (standard **output**).

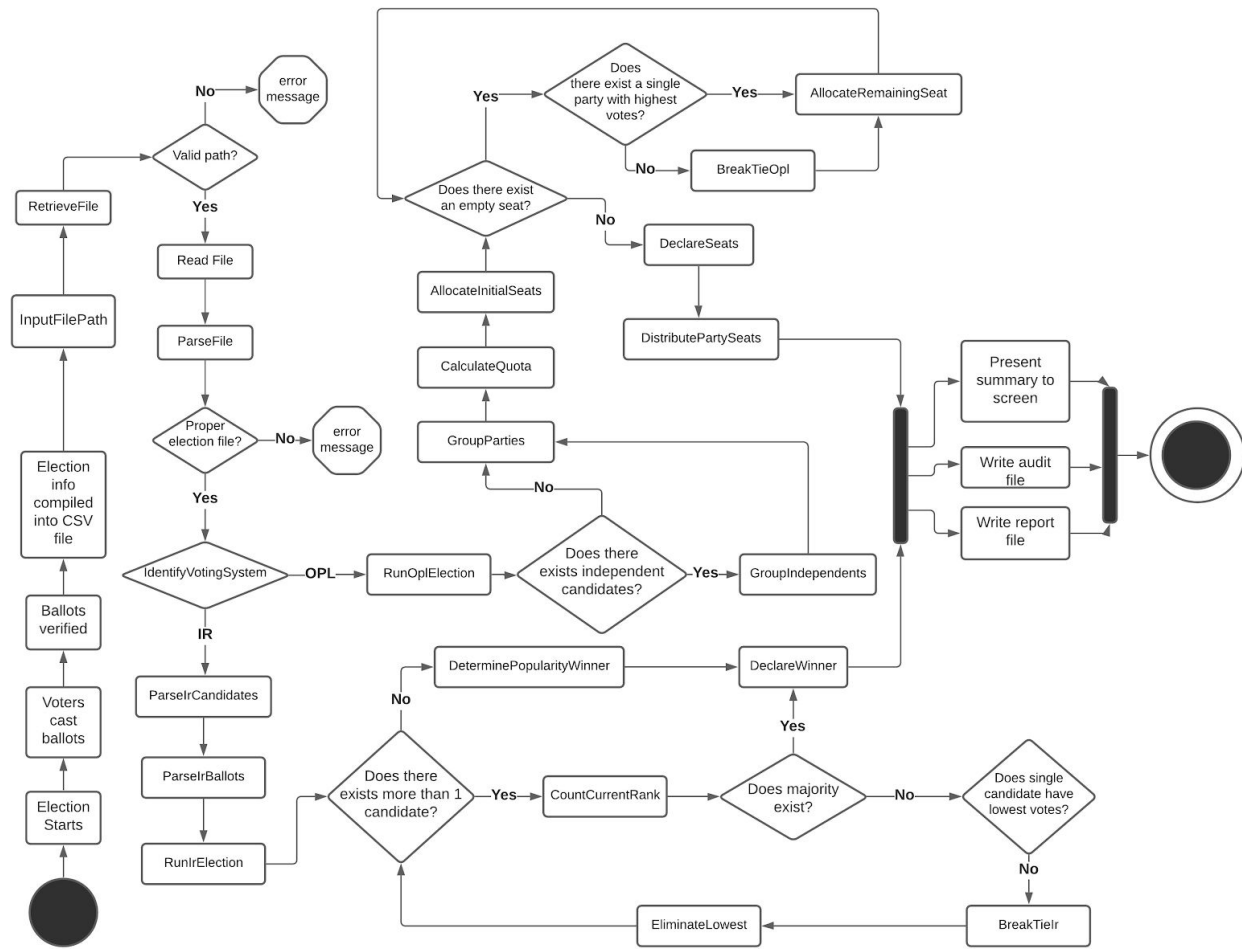
### 3.4 Communications Interfaces

*<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.>*

The only communication that CompuVote does includes system calls for reading from and writing to files. It reads (or attempts to read) from the file provided via **command line arguments**. It **outputs** summary information to the **CLI** (standard **output**), to an **audit file** in the `audits` **directory** (which is directly under the `repo-Team19` **directory**), and to a report file in the `reports` **directory** (which is also directly under the `repo-Team19` **directory**).

## 4. 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.>*



The diagram above displays a flow chart that defines the general flow of operations within the CompuVote system.

[Click here](#) to be redirected to the Use Cases.

## 4.1 Input File Location

### 4.1.1 Description and Priority

A user needs to be able to **input** the location to an election **CSV file** into CompuVote.

### 4.1.2 Stimulus/Response Sequences

*<List the sequences of user actions and system responses that stimulate the behavior defined for this feature. These will correspond to the dialog elements associated with use cases.>*

| Stimulus  | Response   |
|---|--|
| The election official user runs the program with one <b>command line argument</b> .                   | CompuVote confirms that only one command-line <b>argument</b> has been received and stores it.   |
| The election official user runs the program with some other number of <b>command line arguments</b> . | CompuVote prints an error message that the program only accepts one <b>command line argument</b> : the <b>absolute file path</b> or the path relative to repo-Team19 to the election file. |

### 4.1.3 Functional Requirements

*<Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present in order for the user to carry out the services provided by the feature, or to execute the use case. Include how the product should respond to anticipated error conditions or invalid **inputs**. Requirements should be concise, complete, unambiguous, verifiable, and necessary. Use “TBD” as a placeholder to indicate when necessary information is not yet available.>*

*<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>*

#### UC\_001: InputFilePath

##### Description:

The user runs the command to start the program and passes in a single command-line **argument**, which is preferably an **absolute path** or path relative to the repo-Team19 to the election file to parse.

##### Exceptions:

1. The user does not **input** exactly one command-line **argument** to the program
  - a. Response: An error is printed to the screen stating that the program only accepts one **command line argument**. Then, the program closes with a nonzero status.

## 4.2 Read and Parse File

### 4.2.1 Description and Priority

The system needs to be able to read in and parse the provided election file for the information required to simulate the running of an election.

### 4.2.2 Stimulus/Response Sequences

*<List the sequences of user actions and system responses that stimulate the behavior defined for this feature. These will correspond to the dialog elements associated with use cases.>*



| Stimulus  | Response  |
|---|---|
| The user has <b>inputted</b> one <b>command line argument</b> .   | CompuVote assumes that the provided <b>argument</b> is an <b>absolute file path</b> or a <b>file path</b> relative to <code>repo-Team19</code> and attempts to retrieve an <b>input stream</b> .  |
| The provided <b>command line argument</b> was not a valid <b>absolute file path</b> or a <b>file path</b> relative to <code>repo-Team19</code> .  | CompuVote prints an error message stating that a file could not be found at the given <b>file path</b> and exits with a nonzero status.   |
| The provided <b>command line argument</b> was a valid <b>file path</b> , but the program does not have the permissions required to read the file. | CompuVote prints an error message stating that the file at the given <b>file path</b> could not be read from and exits with a nonzero status.   |
| CompuVote is successfully able to get an <b>input stream</b> to the <b>file path</b> provided.  | CompuVote begins to parse the file, starting with the first line where the election type should be stored.  |
| The file is in the format for an <b>IR</b> election specified in subsection 5.5.  | CompuVote parses the election type as instant run-off voting and returns the voting system instance associated with <b>IR</b> .   |
| The file is in the format for an <b>OPL</b> election specified in subsection 5.5.   | CompuVote parses the election type as <b>OPL</b> voting and returns the voting system instance associated with <b>OPL</b> .   |
| The file does not completely match either of the formatting standards for <b>IR</b> or <b>OPL</b> .   | CompuVote prints an error message corresponding to the line on which the discrepancy in format is found in addition to the line number at which the error occurred. It then exits with a nonzero status. (See the use cases that begin with “Parse” without quotes to see a more detailed explanation of this.) |

#### 4.2.3 Functional Requirements

*<Itemize the detailed functional requirements associated with this feature. These are the software capabilities that must be present in order for the user to carry out the services provided by the feature, or to execute the use case. Include how the product should respond to anticipated error conditions or invalid inputs. Requirements should be concise, complete, unambiguous, verifiable, and necessary. Use “TBD” as a placeholder to indicate when necessary information is not yet available.>*

*<Each requirement should be uniquely identified with a sequence number or a meaningful tag of some kind.>*

#### UC\_002: RetrieveFile

**Description:**

The system needs to be able to retrieve and read from the file that the user provides in UC\_001.

**Exceptions:**

1. The **file path** is invalid as it is not a valid **absolute path** to a file or valid path to a file relative to `repo-Team19`.
  - a. Response: The program prints out an error message stating that the file could not be found at the provided **file path**. Then, the program is exited with a nonzero status.
2. The **file path** was valid, but the system cannot read from the file.
  - a. Response: The program prints out an error message stating that the file was found, but CompuVote was unable to read from it. Then, the program exits with a nonzero status.

#### UC\_003: ParseFile

**Description:**

The system needs to be able to parse the provided election file for the information required to simulate the running of an election.

**Exceptions:**

1. There is an issue in parsing any of the subordinate use cases.
  - a. Response: The program prints out an error message, which is the same as the error message printed in the corresponding subordinate use case. Then, the program exits with a nonzero status.

#### UC\_004: IdentifyVotingSystem

**Description:**

The system needs to be able to parse the provided election file for the election type.

**Exceptions:**

1. The election type is not **"IR"** or **"OPL"** without quotes.
  - a. Response: The program prints out an error message stating that the election file's provided voting type does not correspond with the available voting types (**IR** and **OPL**) and the line number where the issue occurred. Then, the program exits with a nonzero status.

## UC\_005: ParseIrCandidates

**Description:**

The system needs to be able to parse the provided election file for the candidates in the **IR** election type.

**Exceptions:**

1. The value provided for the number of candidates is not an **integer**.
  - a. Response: The program prints out an error message stating that the election file's provided value for the number of candidates was not an **integer** and the line number where the issue occurred. Then, the program exits with a nonzero status.
2. The **integer** provided for the number of candidates is negative.
  - a. Response: The program prints out an error message stating that the election file's provided value for the number of candidates was not nonnegative and the line number where the issue occurred. Then, the program exits with a nonzero status.
3. The format of the candidates is not as specified in subsection 5.5 for an **IR** election.
  - a. Response: The program prints out an error message stating that the format of the candidates did not meet what is expected and the line number where the issue occurred. Then, the program exits with a nonzero status.
4. The number of candidates parsed is not equivalent to the number of candidates provided on line 2.
  - a. Response: The program prints out an error that the number of candidates did not match the expected number provided and the line number where the issue occurred. Then, the program exits with a nonzero status.

## UC\_006: ParseIrBallots

**Description:**

The system needs to be able to parse the provided election file for the ballots in the **IR** election type.

**Exceptions:**

1. The value provided for the number of ballots is not an **integer**.
  - a. Response: The program prints out an error message stating that the election file's provided value for the number of ballots was not an **integer** and the line number where the issue occurred. Then, the program exits with a nonzero status.
2. The **integer** provided for the number of ballots is negative.

- a. Response: The program prints out an error message stating that the election file's provided value for the number of ballots was not nonnegative and the line number where the issue occurred. Then, the program exits with a nonzero status.
3. The number of ballots provided does not match the number of ballots provided on line 4.
  - a. Response: The program prints out an error message stating that the election file's provided number of ballots does not match the number of ballots that it said it would give as **input** and the line number where the issue occurred. Then, the program exits with a nonzero status.
4. The ballot provided does not meet the format specified in subsection 5.5 for an **IR** election.
  - a. Response: The program prints out an error message stating that the format of the ballot was not as required and the line number where the issue occurred. Then, the program exits with a nonzero status.

#### UC\_007: ParseOplCandidates

##### **Description:**

The system needs to be able to parse the provided election file for the candidates in the **OPL** election type.

##### **Exceptions:**

1. The value provided for the number of candidates is not an **integer**.
  - a. Response: The program prints out an error message stating that the election file's provided value for the number of candidates was not an **integer** and the line number where the issue occurred. Then, the program exits with a nonzero status.
2. The **integer** provided for the number of candidates is negative.
  - a. Response: The program prints out an error message stating that the election file's provided value for the number of candidates was not nonnegative and the line number where the issue occurred. Then, the program exits with a nonzero status.
3. The format of the candidates is not as specified in section 5.5 for an **OPL** election.
  - a. Response: The program prints out an error message stating that the format of the candidates did not meet what is expected and the line number where the issue occurred. Then, the program exits with a nonzero status.
4. The number of candidates parsed is not equivalent to the number of candidates provided on line 2.

- a. Response: The program prints out an error that the number of candidates did not match the expected number provided and the line number where the issue occurred. Then, the program exits with a nonzero status.

#### UC\_008: ParseNumberOfSeats

**Description:**

The system needs to be able to parse the number of seats for the **OPL** election.

**Exceptions:**

1. The value provided for the number of seats is not an **integer**.
  - a. Response: The program prints out an error message stating that the election file's provided value for the number of seats was not an **integer** and the line number where the issue occurred. Then, the program exits with a nonzero status.
2. The **integer** provided for the number of seats is negative.
  - a. Response: The program prints out an error message stating that the election file's provided value for the number of seats was not nonnegative and the line number where the issue occurred. Then, the program exits with a nonzero status.

#### UC\_009: ParseOptBallots

**Description:**

The system needs to be able to parse the provided election file for the ballots in the **OPL** election type.

**Exceptions:**

1. The value provided for the number of ballots is not an **integer**.
  - a. Response: The program prints out an error message stating that the election file's provided value for the number of ballots was not an **integer** and the line number where the issue occurred. Then, the program exits with a nonzero status.
2. The **integer** provided for the number of ballots is negative.
  - a. Response: The program prints out an error message stating that the election file's provided value for the number of ballots was not nonnegative and the line number where the issue occurred. Then, the program exits with a nonzero status.
3. The number of ballots provided does not match the number of ballots provided on line 5.
  - a. Response: The program prints out an error message stating that the election file's provided number of ballots

- does not match the number of ballots that it said it would give as **input** and the line number where the issue occurred. Then, the program exits with a nonzero status.
4. The ballot provided does not meet the format specified in section 5.5 for an **OPL** election.
    - a. The program prints out an error message stating that the format of the ballot was not as required and the line number where the issue occurred. Then, the program exits with a nonzero status.

## 4.3 Run IR Election

### 4.3.1 Description/Priority

Performs the IR election **algorithms** to process ballots and declare a winner.

### 4.3.2 Stimulus/Response Sequences

| Stimulus   | Response   |
|--|--|
| Parsing of IR election file completes without exceptions   | All candidates' ballots are distributed  |
| The <b>algorithm</b> ranks the candidates.   | <ol style="list-style-type: none"> <li>1. The candidate with the largest vote count is identified</li> <li>2. The candidate with the minimum vote count is identified</li> </ol>             |
| A candidate with majority votes is identified. That is, the candidate that has over 50% of the votes | The candidate that has the majority of votes is declared the winner.   |
| No majority exists among candidates  | A single candidate with the lowest votes is selected for elimination. Their ballots are transferred if the next preference is indicated. Otherwise, the ballot is removed from consideration |
| There are multiple candidates with the minimum vote count.   | The candidate to pick is randomly chosen from the candidates with the minimum vote count   |

### 4.3.3 Functional Requirements

## UC\_011: CountCurrentRank

**Description:**

Identifies the candidate with majority votes and the candidate(s) with the lowest ballot count if there is no candidate with over 50% of total votes.

**Exception:**

1. There are no candidates to eliminate
  - a. Winner is now chosen by popularity instead of a majority

## UC\_12: EliminateLowest

**Description:**

Eliminates the lowest ranked candidate chosen.

## UC\_13: DeterminePopularityWinner

**Description:**

In the event that all eliminations possible have taken place and no candidate has a majority, the candidate with the most votes wins.

## UC\_14: TransferBallot

**Description:**

Transfers eliminated candidate's ballots to the next ranked candidate

## UC\_15: BreakTieIr

**Description:**

Randomly picks a loser when 2 or more candidates have the same number of votes.

## UC\_16: DeclareWinner

**Description:**

Declares a final winner of the IR election. Completes the IR election process.

## 4.4 Run OPL Election

### 4.4.1 Description/Priority

Performs the OPL election **algorithms** to process ballots and declare seats.

### 4.4.2 Stimulus/Response Sequences

| Stimulus | Response |
|----------|----------|
|----------|----------|

|   |   |
|---|---|
| Parsing of OPL election file completes without exceptions without exceptions  | The <b>algorithm</b> groups each party's candidates together (treating independents as a party) |
| Based on the total number of votes per party divided by the quota, the initial allocation of seats takes place.     | The remaining seats are allocated.  |
| Each party distributes their seats to their top candidates.   | The seats are declared.   |
| Potential ties in the second allocation of seats or potential ties when a party is distributing seats to candidates | The ties are broken randomly  |

#### 4.4.3 Functional Requirements

##### UC\_018: GroupIndependents

**Description:**

Groups all independent candidates together into one party.

##### UC\_019: GroupParties

**Description:**

If it is determined that OPL is the voting system, the program must group all members of the same party together. UC\_018 is essentially a specially-identified result of this use case where all independents are grouped into a single party.

##### UC\_020: CalculateQuotas

**Description:**

The quota is used to determine how many the ratio of votes a party must have for every seat they get in the initial allocation.

##### UC\_021: AllocateInitialSeats

**Description:**

Using the calculated quota, the number of initial seats is calculated for each party by dividing the number of votes per party by the quota (truncating the number of seats).

##### UC\_022: AllocateRemainingSeats

**Description:**

The remainders determined in UC\_021 are compared to each other. A second allocation of seats will occur if there are seats yet to be



allocated, where the next subsequent party with the next most votes is granted the seat.

#### UC\_023: BreakTieOpl

##### **Description:**

Breaks a tie in allocating remaining seats in UC\_022 if two or more candidates have the same number of seats and breaks a tie in choosing which candidate gets a seat for a party in UC\_025 if they have the same number of votes in the party.

#### UC\_024: DeclareSeats

##### **Description:**

The seats allocated to parties and candidates will be declared.

#### UC\_025: DistributePartySeats

##### **Description:**

Distributes the seats for each party to the candidates with the highest votes in order

## 4.5 Produce Audit File

### 4.5.1 Description/Priority

The system must be able to produce an **audit file** that contains election information. This file will be useful for backend **software** developers and election officials, with rather **raw data** of the **algorithm** that took place.

### 4.5.2 Stimulus/Response Sequences

| Stimulus   | Response  |
|--|---|
| IR or OPL election <b>algorithm</b> successfully completes | <ol style="list-style-type: none"> <li>Creates a name for the <b>audit file</b> in the format<br/> “audit_&lt;year&gt;-&lt;month&gt;-&lt;day&gt;_&lt;hours&gt;-&lt;minutes&gt;-&lt;seconds&gt;.txt” without quotes where the time is in 24-hour time. Additionally, months, days, hours, minutes, and seconds are all padded by 0s such that they all have 2 digits at all times.<br/> &lt;year&gt;, &lt;month&gt;, &lt;day&gt;, &lt;hours&gt;, &lt;minutes&gt;, and</li> </ol> |

|   |  |
|---|--|
|   | <p>&lt;seconds&gt; are all replaced with the details corresponding to the current timestamp.</p> <ol style="list-style-type: none"> <li>Creates the file with the format mentioned above</li> <li>Writes the audit information to the file.</li> </ol> |
| Program can not create the file in the <code>audits</code> <b>directory</b> | <ol style="list-style-type: none"> <li>The program prints an error stating that the program could not write to the <code>audits</code> <b>directory</b>.</li> <li>The program exits with a nonzero status</li> </ol>                                   |

#### 4.5.3 Functional Requirements

##### UC\_026: WriteAuditToFile

###### **Description:**

An **audit file** will be produced upon completion of the program. This will include the following:

- Gives in-detail explanation of all the steps that occurred in the election.
- Tallying of each individual ballot
- Type of election
- Number of candidates
- The candidate and their parties
- Number of ballots
- Number of seats (where applicable)
- The quota (where applicable)
- Every calculation that occurs during the election
- How many votes each candidate had
- The percentage of votes each candidate had

- **Sample IR input CSV file:**

```
IR
4
Rosen (D), Kleinberg (R), Chou (I), Royce (L)
6
1,3,4,2
1,,2,
1,2,3,
3,2,1,4
,,1,2
,,,1
```

- **Sample IR audit file:**

- <https://pastebin.com/za0DpV8K>

- Sample OPL input **CSV file**:

```
OPL
6
[Pike,D], [Foster,D],[Deutsch,R], [Borg,R], [Jones,R],[Smith,I]
3
9
1,,,,,
1,,,,,
,1,,,,,
,,,1,
,,,,,1
,,,1,,
,,,1,,
,,,1,,
1,,,,,
,1,,,,,
```

- Sample OPL **audit file**:
  - <https://pastebin.com/eWMHuTvK>

## 4.6 Produce Report File

### 4.6.1 Description/Priority

The system must be able to produce a formatted report that contains election information. This file is meant to be understood by anybody who reads it, and so this is the file that would be sent out for media purposes.

### 4.6.2 Stimulus/Response Sequences

| Stimulus   | Response  |
|--|---|
| IR or OPL election <b>algorithm</b> successfully completes | <p>4. Creates a name for the <b>audit file</b> in the format<br/>           “report_&lt;year&gt;-&lt;month&gt;-&lt;day&gt;_&lt;hours&gt;-&lt;minutes&gt;-&lt;seconds&gt;.txt”<br/>           without quotes where the time is in 24-hour time.<br/>           Additionally, months, days, hours, minutes, and seconds are all padded by 0s such that they all have 2 digits at all times. &lt;year&gt;, &lt;month&gt;, &lt;day&gt;, &lt;hours&gt;, &lt;minutes&gt;, and &lt;seconds&gt; are all replaced with the details</p> |

|  |   |
|--|---|
|  | corresponding to the current timestamp.<br>5. Creates the file with the format mentioned above<br>6. Writes the report information to the file.                     |
| Program can not create the file in the <code>reports</code> <b>directory</b> | 3. The program prints an error stating that the program could not write to the <code>audits</code> <b>directory</b> .<br>4. The program exits with a nonzero status |

#### 4.6.3 Functional Requirements

##### UC\_027: PresentReportToScreen

###### **Description:**

A formatted report will be produced upon completion of the program. This will include the following:

1. Type of election
2. Number of candidates
3. The candidate and their parties
4. Number of ballots
5. Number of seats (only for OPL)
6. The quota (only for OPL)
7. How many votes each candidate had

- **Sample IR input CSV file**

```

IR
4
Rosen (D), Kleinberg (R), Chou (I), Royce (L)
6
1,3,4,2
1,,2,
1,2,3,
3,2,1,4
,,1,2
,,,1

```

- 
- **Sample IR report file:**
  - <https://pastebin.com/StX3Et9P>
- **Sample OPL input CSV file:**

```

OPL
6
[Pike,D], [Foster,D],[Deutsch,R], [Borg,R], [Jones,R],[Smith,I]
3
9
1,,,,,
1,,,,,
1,,,,,
,,,1,
,,,1,
,,,1
,,,1,,
,,,1,,
1,,,,,
1,,,,,

```

- 
- Sample OPL report file:
  - <https://pastebin.com/19rdgmgx>

## 4.7 Display Summary to Screen

### 4.7.1 Description/Priority

The system must be able to display a summary of the election to the screen. The summary is simply meant to let the user know the election has been successfully completed, as well as display important election information.

### 4.7.2 Stimulus/Response Sequences

| Stimulus   | Response  |
|--|---|
| IR or OPL election <b>algorithm</b> successfully completes | A summary of the election with the contents mentioned in UC_025 below is printed to the <b>CLI</b> (standard <b>output</b> ). |
|  |   |

### 4.7.3 Functional Requirements

#### UC\_025: PresentSummaryToScreen

##### Description:

A summary of the election will be printed to the screen upon completion of the entire program. This will include the following:

1. Type of election
2. Number of candidates
3. The candidates and their parties
4. Number of ballots
5. Number of seats (only for OPL)

6. The quota (only for OPL)
7. How many votes each candidate had

- Sample IR input **CSV file**

```
IR
4
Rosen (D), Kleinberg (R), Chou (I), Royce (L)
6
1,3,4,2
1,,2,
1,2,3,
3,2,1,4
,,1,2
,,,1
```

- 
- Sample IR summary **output**:
  - <https://pastebin.com/StX3Et9P>
- Sample OPL input **CSV file**

```
OPL
6
[Pike,D], [Foster,D],[Deutsch,R], [Borg,R], [Jones,R],[Smith,I]
3
9
1,,,,
1,,,,
,1,,,,
,,,1,
,,,1
,,,1,,
,,,1,,
1,,,,
,1,,,,
```

- 
- Sample OPL summary **output**:
  - <https://pastebin.com/shbcRqhM>

## 5. Other Nonfunctional Requirements

### 5.1 Performance Requirements

*<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.>*

CompuVote is made to handle very large elections, and so we need a threshold that is reasonable for elections of any size. This threshold is the ability to run an election of 100,000 ballots in under 8 minutes. Additionally, the voting **algorithm** can be used multiple times a year in different election settings (normal elections and special elections.) CompuVote wants to be as available as possible,

and applying constraints to when it can be used would defeat the purpose of the easy-to-use voting system.

## 5.2 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.>*

There are not any safety constraints that exist for CompuVote. Possible loss damage or harm from use of the system is entirely out of CompuVote's scope of use. The data of the candidates and voters is not used for any purpose other than for producing the **audit file**, report file, and summary to screen.

## 5.3 Security Requirements

*<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.>*

There are not any security constraints that exist. This program is meant to be used by anyone who wants to run an election. CompuVote is not responsible for data management, and no data personal to the voter and candidate is being tracked. The **CSV file** contains all the necessary **inputs** to run an election.

## 5.4 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.>*

- **Adaptability/Flexibility:** CompuVote can only process **CSV files** with specific formatting for instant run-off and open party list voting. However, within the **CSV files**, there are no restrictions on the names of parties or candidates involved within the election as long as the election type is **IRV** or **OPL**, and the file has the required format. Additionally, there is no restriction on the number of candidates or number of ballots other than physical and logical constraints. For reference, 100,000 ballots can be processed under 8 minutes.
- **Testability/Correctness:** Every possible scenario might be difficult to test, but CompuVote can easily be tested for its correctness by creating test **CSV files** to run simulations of elections with outcomes that have already been determined. The results can then be compared to the expected results. Additionally, many of the program's functions will be thoroughly **unit tested** using **JUnit**. See 2.7 for constraints regarding **unit tests** developed for the project.

- Reliability: CompuVote will always process a **CSV file** deterministically; however, the election results might differ if there happens to be ties between any of the candidates. Whenever there is a tie, the program randomly breaks the tie. In such cases, the CompuVote could declare different results when running a **CSV file** multiple times.
- Portability: CompuVote is lightweight and built using Java which provides source code portability. For more information on the operating environment, see section 2.4
- Usability: CompuVote is relatively easy to execute and use, requiring basic knowledge of the **command line**. See section 3 (External Interface Requirements) for more information about how to execute the program.
- Robustness: CompuVote can process 100,000 ballots within 8 minutes.

## 5.5 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.>*

- **IRV** process
  - Voters rank candidates in order of preference
    - i. The mark “1” for their most preferred candidate, and “2” for their second preference and so on
  - First, all the number 1 preferences of the voters are counted.
  - If a candidate receives over 50% of the first choice votes, the candidate is declared elected.
  - If no candidate receives a majority, then the candidate with the fewest votes is eliminated. The ballots of the supporters of this defeated candidate are then transferred to whichever of the remaining candidates they marked as their number two choice.
    - i. If a ballot does not give their next preference, it is removed from the election.
  - After the transfer, the votes are then recounted to see if any candidate now receives a majority of the vote.
  - The process of eliminating the lowest candidate and transferring their votes continues until one candidate receives a majority and wins the election.
  - If there is no clear majority by the end, then popularity wins.
- **OPL** process
  - To determine the number of seats for each party, all votes for each party are added up. Independents are treated as one party
  - Then, a quota number is calculated by taking the total number of valid votes and dividing this by the number of seats
  - The quota is then divided into the vote that each party receives and the party wins one seat for each whole number produced.
    - i. Candidates are given seats based on votes they received
  - After the first allocation of seats is complete, then the remainder numbers for the



parties are compared, and the parties with the largest remainders are allocated the remaining seats

- **CSV file rules**

- Both IRV and OPL ballots do not contain write-in candidates
- **IRV** election
  - i. A first line consisting of “**IR**”
  - ii. A second line consisting of a nonnegative number of candidates in the **IR** election
  - iii. A third line consisting of the number of candidates specified in ii. Each candidate is separated using a comma delimiter and potentially whitespace, and each candidate is in the form “<candidateName> (<party>)” where <candidateName> consists of any characters (except whitespace at the beginning and end) and <party> can be any string (without whitespace at the beginning or end).
  - iv. A fourth line consisting of a nonnegative number of ballots for the **IR** election.
  - v. A number of lines equivalent in size to the number provided in iv. Each line consists of comma-separated values, the number provided being equivalent to the number provided in ii (each corresponding to each candidate in the provided order). Each of the comma-separated values is either a number between 1 and the number of candidates (inclusive) or is empty. A line must have at least one ranked candidate, and the ranking values must be 1 to the number of candidates chosen to rank without any gaps.
- Sample **IR CSV file** format:

```
IR
4
Rosen (D), Kleinberg (R), Chou (I), Royce (L)
6
1,3,4,2
1,,2,
1,2,3,
3,2,1,4
,,1,2
,,,1
```

- 
- **OPL** election
  - i. A first line consisting of “**OPL**”
  - ii. A second line consisting of a nonnegative number of candidates in the **OPL** election
  - iii. A third line consisting of the number of candidates specified in ii. Each candidate is separated using a comma delimiter and potentially whitespace, and each candidate is in the form “[<candidateName>, <party>]” where <candidateName> consists of any characters (except whitespace at the beginning and end) and <party> can be any string (without whitespace at the beginning or end).
  - iv. A fourth line consisting of a nonnegative number of seats for the **OPL** election.

- v. A fifth line consisting of a nonnegative number of ballots for the **OPL** election.
- vi. A number of lines equivalent in size to the number provided in iv. Each line consists of comma-separated values, the number provided being equivalent to the number provided in ii (each corresponding to each candidate in the provided order). Each of the comma-separated values is either 1 or is empty. A line must have only one ranked candidate, denoted by 1.
- Candidate and party names in both **IR** and **OPL** files must be one word.
- Sample **OPL CSV file** format:

```
OPL
6
[Pike,D], [Foster,D],[Deutsch,R], [Borg,R], [Jones,R],[Smith,I]
3
9
1,,,,,
1,,,,,
,1,,,
,,,1,
,,,1
,,,1,,
,,,1,,
1,,,,
,1,,,,
```

- **Command Line** rules

- The data coming into the CompuVote system is the filepath of the election file. CompuVote only accepts one election file. This can be an **absolute path** (that is, relative to `C:\` or another **disk drive** on **Windows** or `/` on **macOS** and **Ubuntu**) or a path relative to the `repo-Team19` **directory**. (Refer to <https://www.computerhope.com/jargon/p/path.htm> for more information on what **file paths** are and how they work.)
- For more information on the requirements for the interface see section 3 (External Interface Requirements)

## 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

*<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.>*

|                      |  |
|----------------------|--|
| <b>32-bit System</b> | Processor that can access $2^{32}$ memory addresses. |
|----------------------|--|

|  |  |
|--|--|
| <b>64-bit System</b>                     | Processor that can access $2^{64}$ memory addresses.   |
| <b>Absolute Path</b>                     | The complete path beginning from the root <b>directory</b> that specifies the location of a <b>directory</b> or file. (C : \ for <b>Windows</b> and / for <b>Ubuntu</b> and <b>macOS</b> )   |
| <b>Algorithm</b>                         | Finite sequence in the form of instructions that solve a class of problems to perform computations.  |
| <b>Argument</b>                          | The <b>input</b> that is passed into the parameters of a function.   |
| <b>Audit File</b>                        | An <b>output</b> file that will be useful to election officials and <b>software</b> developers as it details every step of how a process occurred.   |
| <b>Citrix Receiver</b>                   | A workspace <b>software</b> that allows for usage of <b>Windows</b> workspace on a remote client through Citrix servers.   |
| <b>Command Line Interface (CLI)</b>      | An interface that allows the user to type direct commands to the system in the form of lines of text.  |
| <b>Comma Separated Values File (CSV)</b> | A CSV file usually consists of comma-delimited values separated into groups by newline characters. The <b>input</b> file to the election that will grant all the necessary <b>inputs</b> to run an election. An election cannot be run without the <b>input</b> of a CSV file. |
| <b>Directory</b>                         | A special type of file that contains a list of objects, which can be in the form of files, more directories, or links.   |
| <b>Disk Drive</b>                        | A device that reads and writes data to a disk. (e.g. hard drive, floppy disk).   |
| <b>Duo</b>                               | An authentication service needed to authenticate with <b>VOLE</b> , <b>SSH</b> , <b>Citrix Receiver</b> , and many of the University of Minnesota – Twin Cities' services.   |
| <b>File Path</b>                         | A location of a file or folder within a computer's file system.  |
| <b>Hardware</b>                          | The physical parts of a computer.  |

|  |  |
|--|--|
| <b>Input/Output (I/O)</b>                  | Communication between a system that can process data and a form of external <b>input</b> . For example, a CSV file is an <b>input</b> to CompuVote while the summary printed to the screen and the audit and report files are outputs.   |
| <b>Input</b>                               | Data or information that is entered into a system  |
| <b>Integer</b>                             | A rational number that can be negative, positive, or zero and written without any fractional components.   |
| <b>IntelliJ IDEA IDE</b>                   | An integrated development environment for Java computer <b>software</b> . Is the main <b>software</b> used to create CompuVote.  |
| <b>Instant Runoff Voting (IR) or (IRV)</b> | An election system where voters rank candidates in order of preference. Each round, the candidate with the lowest number of votes will be eliminated, and their rankings are redistributed to the remaining candidates. This system will continue until one candidate has over 50% of the vote, and they will be declared the winner.                          |
| <b>Java Development Kit (JDK)</b>          | A development kit includes items that allows Java programs to run and be developed.  |
| <b>JUnit</b>                               | A <b>unit testing</b> framework used for Java.   |
| <b>Kumo</b>                                | A form of cloud storage information that is used in tandem with the <b>Citrix Receiver</b> app.  |
| <b>Linux</b>                               | A Unix-like operating system that was developed by Linus Torvalds. This <b>OS</b> was developed because Linus thought that there should be more available options in Unix, so he created Linux.  |
| <b>macOS</b>                               | An operating system created by Apple that is run on all of their Mac computers.  |
| <b>Open Party Listing (OPL)</b>            | A proportional representation election format where voters select an individual from a list of candidates within a party. There is a quota value that determines which candidates from each party get a seat. A second allocation will occur if there are seats left to be filled, and these will be filled using the truncated votes from the quota division. |

|                                   |   |
|-----------------------------------|---|
| <b>Operating System (OS)</b>      | The system in which <b>software</b> communicates with <b>hardware</b> , and other programs are able to run.   |
| <b>Output</b>                     | Data or information generated from a system. In the case of CompuVote, this will be in the form of a printed summary to the screen, an audit file, and a report file.   |
| <b>Random Access Memory (RAM)</b> | A <b>hardware</b> component in electronic devices in which <b>OS</b> and application data in current use is stored such that it can be reached by the processor faster.   |
| <b>Raw data</b>                   | Unprocessed computer data. No standard format, may just be a collection of numbers or characters or file.   |
| <b>README file</b>                | A file that will contain useful information regarding the <b>software</b> . The README should be read before first use of a program.  |
| <b>Repository</b>                 | A centralized file storage location. Can store multiple versions of files for the purposes of storing and accessing changes.  |
| <b>Software</b>                   | A general term for computer programs. Relative terms are: software programs, applications, scripts, or instructions.  |
| <b>Secure Shell (SSH)</b>         | Method of communicating (securely) with another computer. All data shared will be encrypted, as only the host and the client have access to information. A third party who tries to intercept the message would receive scrambled and unreadable information. |
| <b>Terminal</b>                   | An application that allows commands to be sent into the computer. Many of the commands regard navigation through a <b>directory</b> or the reach a file destination.  |
| <b>Ubuntu</b>                     | A complete <b>Linux OS</b> that is freely available with the community and professional support.  |
| <b>User Interface (UI)</b>        | The means in which a person controls a <b>software</b> application or <b>hardware</b> device.   |
| <b>Unit Testing</b>               | A <b>software</b> testing method in which individualized sets of code are tested. Each functionality of the program should correspond   |

|   |   |
|---|---|
|   | to a unit test.   |
| <b>Virtual Online <u>Linux</u> Environment (VOLE)</b> | A remote University of Minnesota - Twin Cities - College of Science and Engineering <b>Linux</b> desktop that allows remote usage of a standardized <b>Linux</b> System.                                  |
| <b>Windows</b>  | Operating system developed by Microsoft that is one of the most widely-used operating systems used today.   |
| <b>Windows Powershell</b>                             | A task-based command-line shell and scripting language designed especially for system administration. Great tool to assist IT people in automating administrative capabilities of the <b>Windows OS</b> . |

## Appendix B: Analysis Models

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

