

## Software Design Document (SDD)

Software design is a process by which the software requirements are translated into a representation of software components, interfaces, and data necessary for the implementation phase. The SDD shows how the software system will be structured to satisfy the requirements. It is the primary reference for code development and, therefore, it must contain all the information required by a programmer to write code. The SDD is performed in two stages. The first is a preliminary design in which the overall system architecture and data architecture is defined. In the second stage, i.e. the detailed design stage, more detailed data structures are defined and algorithms are developed for the defined architecture.

This document is an annotated outline for a software design document adapted from the IEEE Recommended Practice for Software Design Descriptions. The IEEE Recommended Practice for Software Design Descriptions have been reduced in order to simplify this assignment while still retaining the main components and providing a general idea of a project definition report. For your own information, please refer to [IEEE Std 10161998](http://www.cs.concordia.ca/~ormandj/comp354/2003/Project/ieeeSDD.pdf)<sup>1</sup> for the full IEEE Recommended Practice for Software Design Descriptions.

---

<sup>1</sup> <http://www.cs.concordia.ca/~ormandj/comp354/2003/Project/ieeeSDD.pdf>

Team 4

## **U.S Ballot Voting System**

Software Design Document

Jan Achumbre, Thien Nguyen, Andrew Tran, Jeffrey Chen

Lab Section:  
Workstation

Date: 03/02/2023

## TABLE OF CONTENTS

<b>1.</b>	<b>INTRODUCTION</b>	<b>2</b>
1.1	Purpose	2
1.2	Scope	2
1.3	Overview	2
1.4	Reference Material	2
1.5	Definitions and Acronyms	2
<b>2.</b>	<b>SYSTEM OVERVIEW</b>	<b>2</b>
<b>3.</b>	<b>SYSTEM ARCHITECTURE</b>	<b>2</b>
3.1	Architectural Design	2
3.2	Decomposition Description	3
3.3	Design Rationale	3
<b>4.</b>	<b>DATA DESIGN</b>	<b>3</b>
4.1	Data Description	3
4.2	Data Dictionary	3
<b>5.</b>	<b>COMPONENT DESIGN</b>	<b>3</b>
<b>6.</b>	<b>HUMAN INTERFACE DESIGN</b>	<b>4</b>
6.1	Overview of User Interface	4
6.2	Screen Images	4
6.3	Screen Objects and Actions	4
<b>7.</b>	<b>REQUIREMENTS MATRIX</b>	<b>4</b>
<b>8.</b>	<b>APPENDICES</b>	<b>4</b>

# 1. INTRODUCTION

## 1.1 Purpose

This Software Design Document is intended to display how the U.S Ballot Voting System program works and shows how the inner components relate with each other using C++. The U.S Ballot Voting System is a software program designed to simplify the process of vote counting during elections.

The expected audience for this document are Ballot officials who will be handling the voting during election times and faculty members from the University of Minnesota.

## 1.2 Scope

This contains all of the description and design aspects of the U.S Ballot Voting System. Essentially, the main will act as a controller which calls all of the other functions and instantiates objects of different classes such as the Candidate (Entity) class, or the Audit class.

The goal of this program is to simplify the task of ballot officials, allowing them, with ease, to tally up votes. With time being of great importance, this program greatly benefits the government as the task is not only simplified, but greatly expedited.

The system will be developed in C++ and will be compatible with Windows, Mac, and Linux platforms. The main goal of this project is to provide an efficient and accurate ballot voting system that can be used for a variety of elections.

The program may not be able to handle very large numbers of votes over 9999, or it may only be able to run on computers with certain specifications.

## 1.3 Overview

This document provides an overview of the architecture and design of the ballot voting system. The document is organized into several sections, starting with the system overview, which provides a general description of the system functionality and design. The system architecture section describes the modular program structure of the system, including the major subsystems and data repositories and their interconnections. The data design section describes how the information domain of the system is transformed into data structures, including the major data or system entities that are stored, processed, and organized. The component design section takes a closer look at each component of the system and provides a summary of the algorithms used in each component. The human interface design section describes the user interface and the functionality of the system from the user's perspective. Finally, the requirements matrix section provides a cross-reference that traces components and data structures to the requirements in the SRS document.

## 1.4 Reference Material

## 1.5 Definitions and Acronyms

**IRV:** Instant Runoff Voting, a voting method used to elect a single winner from a field of candidates with preferential voting.

**CPL:** Closed Party Listing, a type of ballot design used in some elections where candidates are listed by political party, and voters select a party rather than individual candidates.

**Interface:** A class that is pure virtual, which means that it has no implementation code of its own but only provides a set of functions that must be implemented by any derived class.

**UML:** Unified Modeling Language, a standardized modeling language used in software engineering to depict the architecture, design, and implementation of software systems.

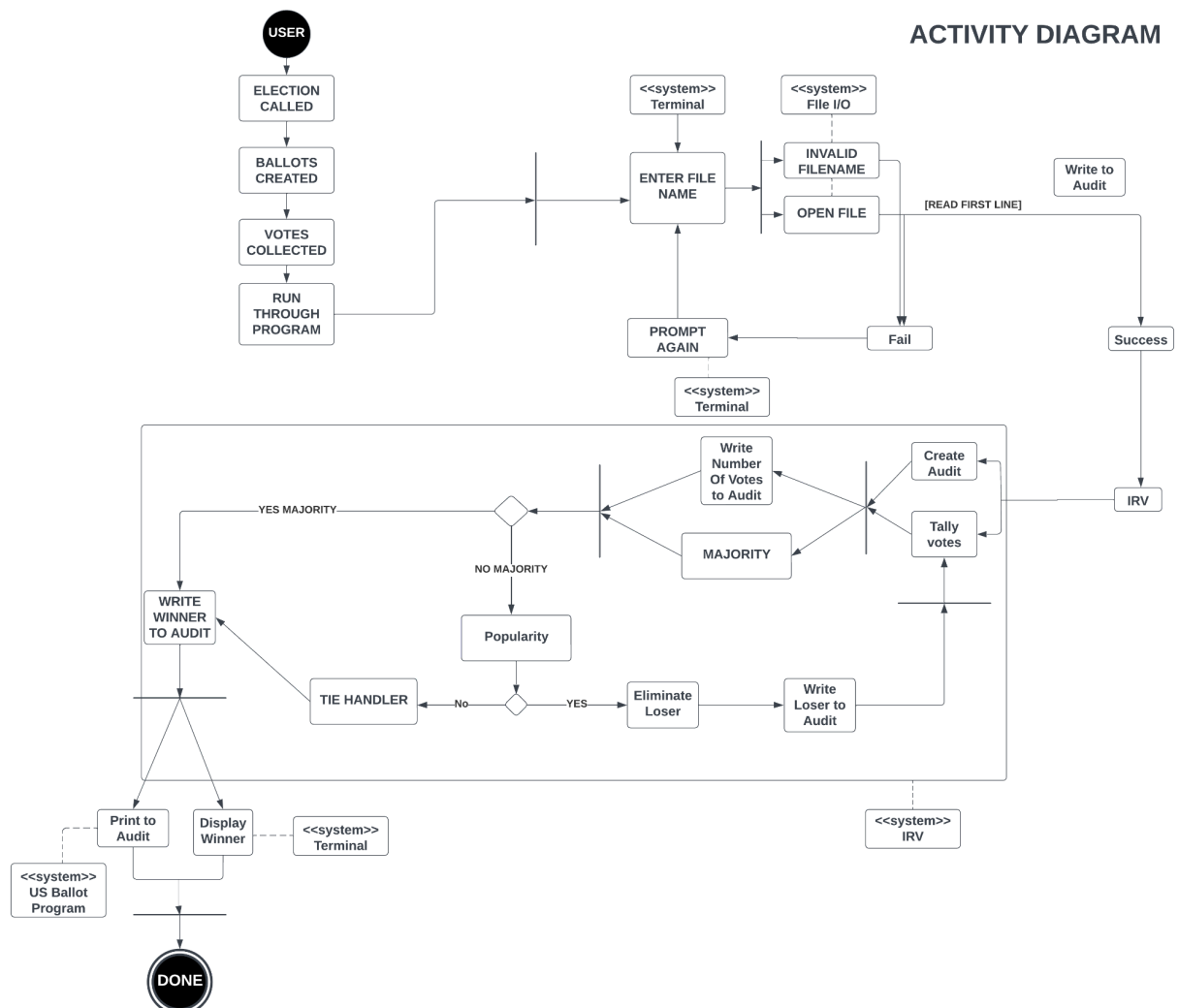
**Candidate class:** A class in the U.S Ballot Voting System program that represents a candidate running for office.

**Ballot class:** A class in the U.S Ballot Voting System program that represents a ballot with the list of candidates and the votes cast for each candidate.

**I/O:** Input/Output

## 2. SYSTEM OVERVIEW

The U.S Ballot Voting System is a computer program that makes it easy for people to vote in elections. Our Ballot Voting System will run 2 types of voting systems: Instant Runoff Voting (IRV) and Closed Party Listing (CPL). IRV helps choose one winner from a collection of candidates, and CPL groups candidates by political party, so people vote for a party instead of individual candidates. The U.S Ballot Voting System is designed to make sure that votes are counted accurately and secretly.



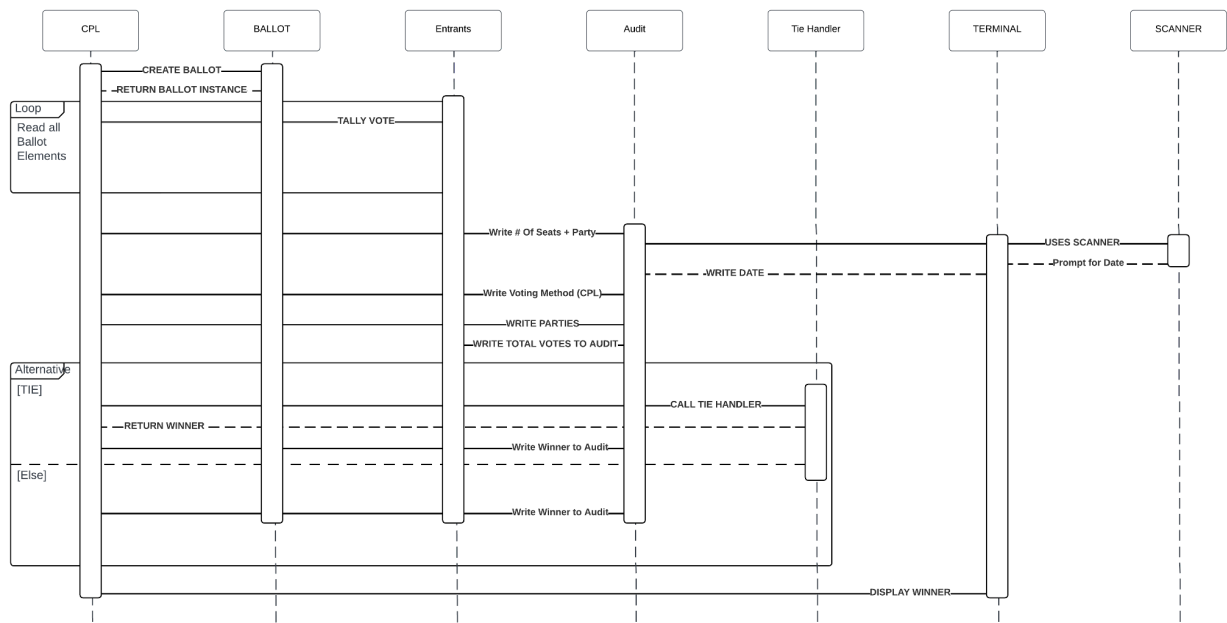
The U.S Ballot Voting System is composed of several subsystems that work together to provide the desired functionality.

The system's input subsystem reads the input file containing the type of voting process, the list of candidates, and other information.

The output subsystem writes the audit file containing the election results.

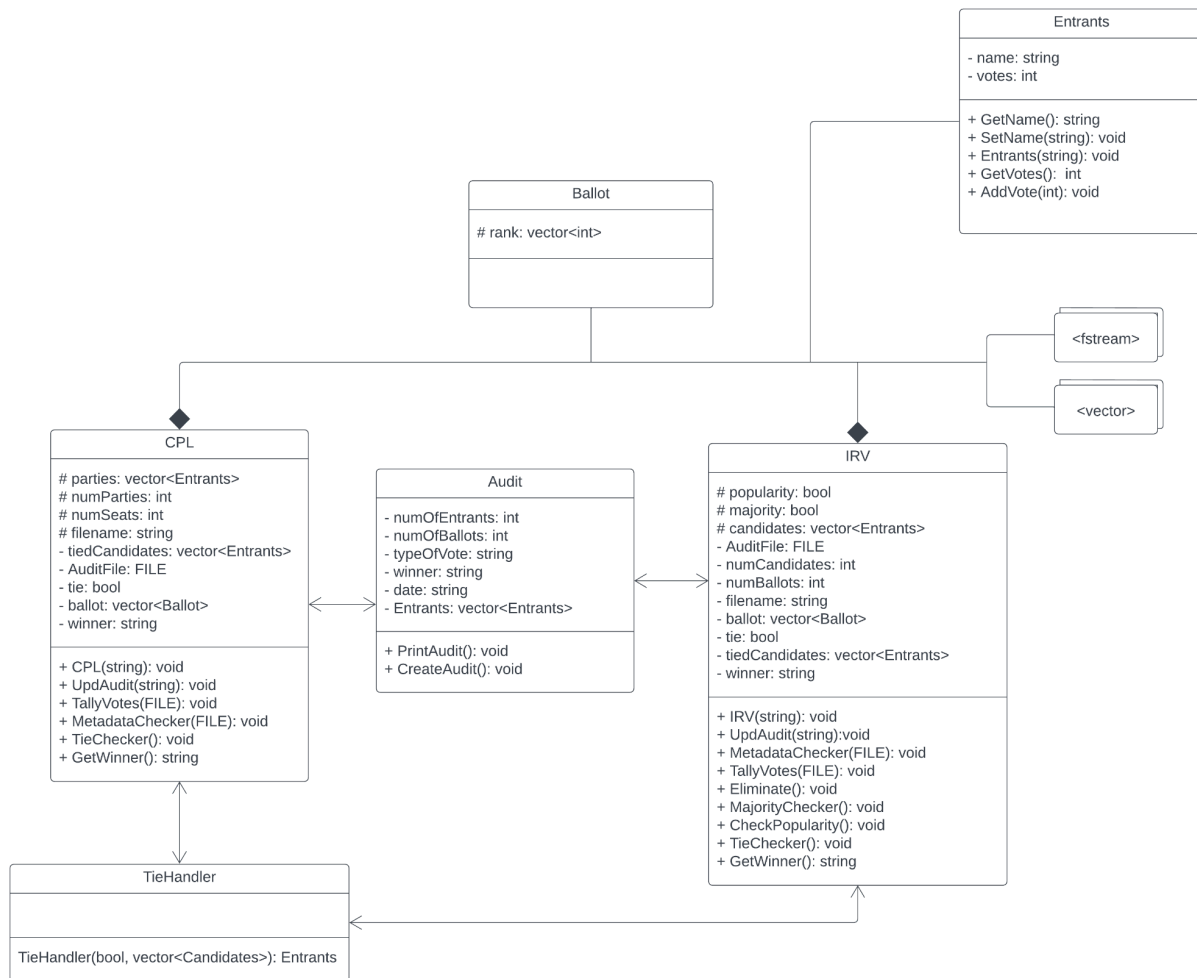
The voting system subsystem is responsible for processing the votes and ensuring that each vote is valid. This subsystem receives the votes from the main controller, validates them, and stores them in the database. It also accumulates the votes and generates the election results.

### 3.2 Decomposition Description



5

### 3.3 Design Rationale



The architecture described in 3.1 was selected based on several critical issues and trade-offs that were considered during the design process. The modular program structure was chosen because it provides a scalable and maintainable solution, which allows for easy modification of the system as the requirements change.

The input and output subsystems were designed to allow for easy reading and writing of data, while the voting system and database subsystems were designed to provide efficient and accurate vote processing and data storage. These subsystems work together to ensure the confidentiality, integrity, and accuracy of the voting process.

Frequently writing to our database can result in a crucial problem of slowing down the program as it frequently makes system calls to the file. It is a significant trade-off since we could have chosen another approach of creating a result variable, constantly appending strings to it, and writing it into the file at once. However, this approach can also affect the program's performance since it involves multiple function calls to append to the string and then write to the file at the end.



## 4. DATA DESIGN

### 4.1 Data Description

The U.S Ballot Voting System transforms the information domain of the system into data structures that can be stored, processed, and organized. The system stores the ballot data in CSV format, which includes the necessary information for each voting process.

For Instant Runoff Voting (IRV), the CSV file contains the number of candidates, the candidates names separated by commas, and the number of ballots in the file. For Closed Party Listing (CPL), the CSV file contains the number of parties, the parties separated by commas, the number of seats, and the number of ballots.

The system uses the Candidate Class to store information about each candidate, including their name, party affiliation, and the number of votes they received. The Ballot Class stores information about each ballot, including the ranks of each candidate on the ballot, the ballot number, and the results for that ballot.

### 4.2 Data Dictionary

**Ballot Class:** A class that stores information about each ballot, including the ranks of each candidate on the ballot, the ballot number, and the results for that ballot.

Attributes:

Ranks: Array of integers representing the ranks of each candidate on the ballot.

Ballot Number: Integer representing the unique identifier for the ballot.

Results: String representing the result of the ballot.

Methods:

getRanks(): Returns an array of integers representing the ranks of each candidate on the ballot.

getBallotNumber(): Returns the unique identifier for the ballot.

setResults(result: String): Sets the result of the ballot to the given string.

**Entity Class:** A class that stores information about each candidate or party, including their name, party, and the number of votes they received.

Attributes:

Name: String representing the name of the candidate.

Party: String representing the party affiliation of the candidate.

Votes: Integer representing the number of votes the candidate received.

Methods:

getName(): Returns the name of the candidate.

getParty(): Returns the party affiliation of the candidate.

getVotes(): Returns the number of votes the candidate received.

addVote(): Adds one vote to the candidate's vote count.

**CSV File:** A file format used to store ballot data in a comma-separated format.

Attributes:

Data: Comma-separated values containing the necessary information for each voting process.

Input Subsystem: A system component responsible for reading the input file containing the type of voting process, the list of candidates, and etc.

Output Subsystem: A system component responsible for writing the audit file containing the election results.

Results Processor: A system component responsible for calculating and generating the final election results.

Voting Subsystem: A system component responsible for processing the votes and ensuring that each vote is valid.

## 5. COMPONENT DESIGN

The input subsystem will consist of the following components:

- File Reader: This component will read the input file and pass the data to the parser component.
- Parser: This component will parse the input data and create the candidate and ballot objects, which will be passed to the voting subsystem.

The voting subsystem will consist of the following components:

- IRV Processor: This component processes the ballots using the Instant Runoff Voting (IRV) method. The IRV Processor uses an algorithm to eliminate candidates with the least number of first preference votes, redistribute their votes to the remaining candidates, and repeat the process until one candidate has a majority of votes.
- CPL Processor: This component processes the ballots using the Closed Party List (CPL) method. The CPL Processor uses an algorithm to allocate the seats to the candidates based on the party and the number of votes received by each party.
- Ballot Processor: This component validates and verifies the ballots before processing them. The Ballot Processor checks and ensures that the ballot contains a valid ranking of candidates.
- Results Processor: This component calculates and generates the final election results. The Results Processor uses the output from the IRV or CPL Processor and generates the output file containing the election results.

The data subsystem will consist of the following components:

- Candidate Class: This class represents the candidate data, including the candidate name and party affiliation, if any.
- Ballot Class: This class represents the ballot data, including the party and the list of candidates ranked in order of preference.
- Candidate List: This class manages the list of candidates and provides methods for adding, removing, and retrieving candidate objects.
- Ballot List: This class manages the list of ballots and provides methods for adding, removing, and retrieving ballot objects.

The output subsystem will consist of the following components:

- Audit file: This component shows the election information at the time, such as the type of voting, number of candidates, candidates, number of ballots, calculations, and how many votes a candidate had. It lists the winner(s) and shows how the election progressed so that the audit could replicate the election itself.
- Results: This component shows the winner of the election without any other information.

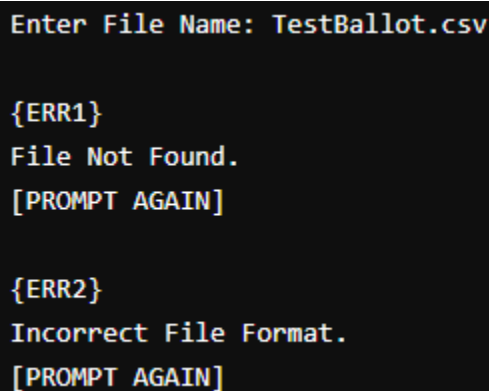
## 6. HUMAN INTERFACE DESIGN

### 6.1 Overview of User Interface

Upon starting the program, the user will be prompted to enter a file name in the terminal. If an error occurs during this time, then the user will be prompted to enter the file again with an appropriate message regarding the specific error. If there are no errors that occur at this time, then the system will run until a specific information is required from the user such as the current date.

Upon completion of the program, then the user will be able to see the results of the election. What they will see consists of the following: The winner (Candidate or Party name), the total votes, and the votes that went towards the winning candidate. In the case of CPL, we will also display the rankings and the votes that went to those said parties.

### 6.2 Screen Images



```
Enter File Name: TestBallot.csv

{ERR1}
File Not Found.
[PROMPT AGAIN]

{ERR2}
Incorrect File Format.
[PROMPT AGAIN]
```

```
## IRV Results
Winner: NAME
Type: Instant Runoff
Total Votes: 9999
Votes Towards Candidate: 9998

## CPL Results
Seat Winners: PARTY1, PARTY2, PARTY3, ...
Type: Closed Party Listing
Total Votes: 9999
PARTY1 Votes: . . .
PARTY2 Votes: . . .
PARTY3 Votes: . . .

{ERR}
Something Went Wrong During Calculation Process.
```

### 6.3 Screen Objects and Actions

In the first section, the user is prompted to enter the file which is done through an instance of a scanner class and prompting the user in the terminal to enter a string, in this case, the file name. With this scanner class, we are able to check if a file exists or not and if it does not, then we display an error message telling the user that the file was not found and prompting them to enter the file again. Using an instance of C++ File I/O, we are able to parse through the file. In the case of the second error displayed in the first section, we parse the first few lines of the file to ensure that it is of the correct format.

In the second section of the images, the user is displayed a result of the election. There are many elements that are in play here, specifically the standard output, the IRV/CPL classes, and the audit class. Putting it simply, the IRV/CPL classes are constantly updating an audit object and the audit (at the end of the program's cycle) prints out the result to the standard out.

## 7. REQUIREMENTS MATRIX

USE CASE	System/Component/Function	Class/Object/Data Structure
1, 9	Terminal - prompt and reading file	fstream, I/O, string
4	IR voting	Ballot, IRV, Entrants, Audit
7, 10, 11	CPL	Ballot, CPL, Entrants, Audit
5	Coin Flip	TieHandler
6	Popularity	TieHandler
3, 8	Terminal - Display winner	I/O
2, 12, 13	Audit file	Audit