

Software Design Document (SDD) Template

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 template 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)¹ for the full IEEE Recommended Practice for Software Design Descriptions.

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

(Team 24) **(Voting System)** Software Design Document

Name (s):

Anthony Ross-Sapienza, ross001

Philip Siedlecki, siedl009

Xiuyu Yan, yanxx401

Jack Levine, levin520

Date: 11/4/2019

TABLE OF CONTENTS

1. INTRODUCTION	4
1.1 Purpose	4
1.2 Scope	4
1.3 Overview	4
1.4 Reference Material	5
1.5 Definitions and Acronyms	5
2. SYSTEM OVERVIEW	6
3. SYSTEM ARCHITECTURE	7
3.1 Architectural Design	7
3.2 Decomposition Description	10
3.3 Design Rationale	10
4. DATA DESIGN	12
4.1 Data Description	12
4.2 Data Dictionary	12
5. COMPONENT DESIGN	16
6. HUMAN INTERFACE DESIGN	23
6.1 Overview of User Interface	23
6.2 Screen Images	23
6.3 Screen Objects and Actions	24
7. REQUIREMENTS MATRIX	25

1. INTRODUCTION

1.1 Purpose

This software design document describes the architecture and system design of the 5801 Voting System. The system has been designed to act as an automated means to tabulate votes and output the results of a file provided election. The system's expected audience is election officials utilizing the system to discern the results of an election.

1.2 Scope

This document outlines the complete design structure of the 5801 Voting System. The basic architecture is a C++ program executed through the UNIX terminal, utilizing C's built in file I/O handling. The system has been designed as a simple, fast, low demand, and, most importantly, accurate means for election officials to determine the results of an election.

1.3 Overview

Section 2 is the System Overview which details an overview of the system's design and how the interworking components work to create the overarching functionality.

Section 3 is the System Architecture detailing a high level description of the system structure and the relationships between interacting components.

Section 4 is the Data Design describing how information is read, transferred, and stored among the system components.

Section 5 is the Component Design outlining the specific functionality of each component within the system.

Section 6 is the Human Interface Design which describes the system's use and functionality from a user's perspective.

Section 7 is the Requirements Matrix detailing the design reasoning for each component with respect to justifications made in the software requirements specification document.

1.4 Reference Material

IEEE. IEEE Std 1016-1998 IEEE Recommended Practice for Software Design Descriptions. IEEE Computer Society, 1998

1.5 Definitions and Acronyms

Term	Definition
OPL	Open Party List
CPL	Closed Party List
CSV	Comma Separated Values

2. SYSTEM OVERVIEW

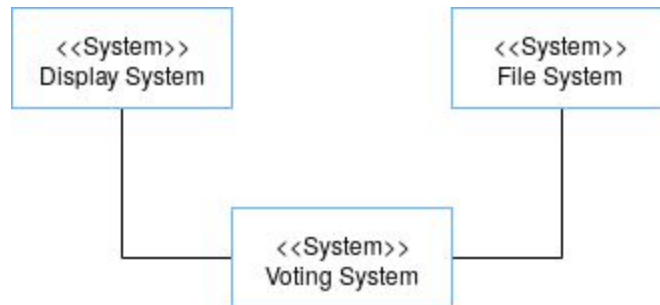


Figure 2.1: Context model for 5801 Voting System.

The 5801 Voting System program is designed to quickly and accurately decide the results of an election based on voting data provided to the program at execution. The provided file is expected to be of CSV format and the file data to be of either OPL or CPL voting type. The system is separated into three core systems (as shown in figure 2.1). The main system, or control system, is the Voting System; here is where program execution starts and ends, calling methods of the File System to read and inform on data contained within the given file. The File System, as loosely described previously, handles the reading, tabulating, tie breaking, and, where needed, file type conversion for the input data. The Display System handles the interaction between the program and the user through the UNIX terminal; here the data is provided to the Voting System and the final results are displayed. Together, the three systems make up the full functionality of the 5801 Voting System program.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

Voting System

Name: Voting_System

Type: Driving class

Description: This is the main class of the voting system, it will drive the main flow of the computation and be responsible for user output.

Attributes: data_ : File_Data

 elected_ : Candidate_Data *

Resources: None

Operations:

Name: Voting_System(char *)

Arguments: name of file containing vote data.

Returns: None

Pre-condition: The file must exist in the same directory, and formatted correctly.

Post-condition: The data_ and elected_ fields will be populated.

Exceptions: File is not in folder, or file is formatted incorrectly.

Flow of Events:

1. Call the File_Data constructor for data_.
2. Fill elected_ with the top (party_list_.seats_ #) candidates from each party.

Name: Voting_System()

Arguments: None

Returns: None

Pre-condition: None

Post-condition: The data_ and elected_ fields will be populated.

Exceptions: File is not in folder, or file is formatted incorrectly.

Flow of Events:

1. Prompt the user for a file to load.
2. Call the File_Data constructor for data_.
3. Fill elected_ with the top (party_list_.seats_ #) candidates from each party.

Name: Audit_File()

Arguments: None

Returns: None

Pre-condition: Voting_System(char*) must have been called.

Post-condition: An audit file will be created within the current directory.

Exceptions: None

Flow of Events:

1. Create an audit file under the current directory.
2. Write all pertinent audit data to the file.

Name: Media_File()

Arguments: None

Returns: None

Pre-conditions: Voting_System(char *) must have been called.

Post-Condition: A Media file will be created within the current directory.

Exceptions: None

Flow of Events:

1. Create a media file under the current directory.
2. Write the elected officials to the media file.

File Data

Name: File_Data

Type: Computational module

Description: The File_Data class is used to read in file data, and return the party data in a cleanly formatted array. This class will handle OPL and CPL, converting the OPL data to a CPL format. The File_Data class also allows viewing of all variables, for audit file generation.

Attributes: voting_type_ : char *

num_parties_ : int

party_list_ : Party_Data *

num_seats_ : int

num_ballots_ : int

num_candidates_ : int

Candidates_ : Candidate_Data *

quota_ : float

Resources: None

Operations:

Name: File_Data(char *)

Arguments: file name (vote data file).

Returns: None

Pre-condition: The file exists.

Post-condition: All fields will be filled in according to the file given.

Exceptions: File invalid

Flow of Events:

1. Call Load_File(char *) with the given file name.

Name: Load_File(char *)

Arguments: file name (vote data file).

Returns: None

Pre-condition: The file exists.

Post-condition: All fields will be filled in according to the file given.

Exceptions: File invalid

Flow of Events:

1. Check that the file exists.
2. Parse the file data into the appropriate fields.
3. Read both formats in as CPL and convert where needed.
4. Generate party data, including how many seats each party has won.

5. Break ties as needed between parties (seat totals).

Name: Break_Tie(int)

Arguments: Array to break tie on (using enum int).

Returns: None

Pre-condition: There must be at least one party in the party_list_ and party_list_.votes_ must be non NULL (for all).

Post-condition: party_list_ will be sorted by rank (with ties settled).

Exceptions: None

Flow of Events:

1. Sort the party_list_ by party_list_.votes_.
2. Scan through all elements of party_list_ (looking for tied vote totals).
3. Rearrange the party_list_ in tied regions (roll an n sided die to determine rank).

Candidate Data

Name Candidate_Data

Type: Data type

Description: Candidate_Data holds relevant data for each candidate.

Attributes: name_ : char *

party_ : char

votes_ : int

party_rank_ : int

Resources: None

Operations: None

Party Data

Name: Party_Data

Type: Data type

Description: Party_Data holds relevant data for each party.

Attributes: name_ : char *

votes : int

seats_ : int

candidates_ : Candidate_Data *

num_Candidates_ : int

Resources: None

Operations: None

3.2 Decomposition Description

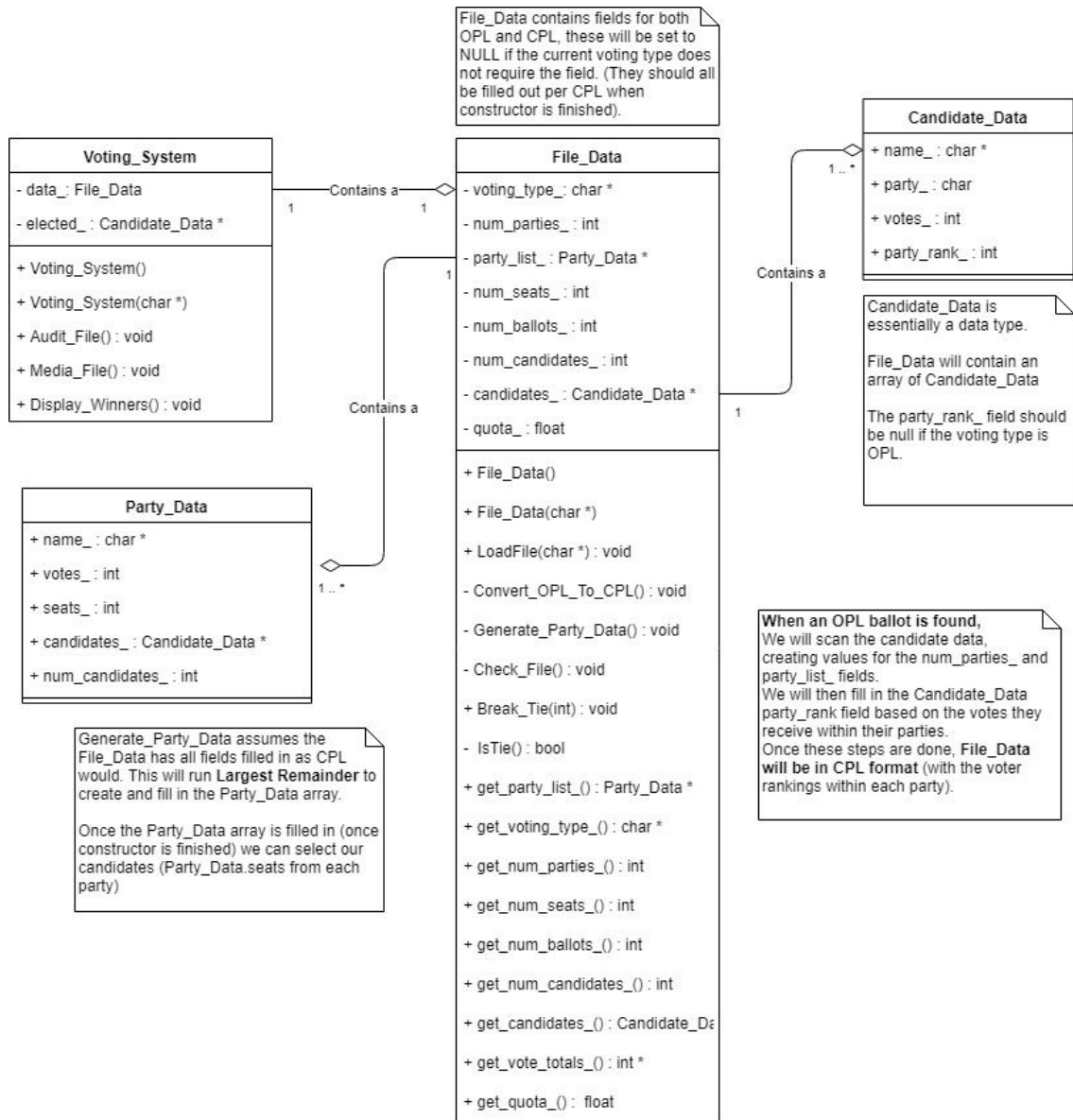


Figure 2.2: Architecture Design

3.3 Design Rationale

The architecture above generally follows the pipeline design pattern. The Voting system lends itself nicely to the pipeline pattern as the user inputs a file, the system then tabulates data, and generates output files. There is no user interaction once the initial file is specified.

With the pipeline design pattern in mind, the system is broken up into several partitions. The data transformation module in `File_Data` and the data analysis portion found in `Voting_System`. If the system were to expand on the data analysis functionality, an additional module would need to be created, but since the current output files are simple, this can be done in the main class.

Creating a separate subsystem that handles data, allows use to take both OPL and CPL files into one class, and output one format that can be used to display the results of both voting types. This shifts the complexity of the system from having two subsystems to handle OPL and CPL (`System_OPL` and `System_CPL`) to a more complex `File_Data` class.

4. DATA DESIGN

4.1 Data Description

The data is read in as a CSV file. The system processes the data and output the voting result to the screen and generates two TXT files: audit file and media file.

Candidate_Data is a class that contains candidate information.

Party_Data is a class that contains party information.

File_Data is a class that contains fields for both OPL and CPL voting types.

Voting_System is a class that contains necessary File_Data and Candidate_Data, it is responsible for processing input file and generating audit file and media file.

4.2 Data Dictionary

Candidate_Data:

Attribute Name	Type	Description
name_	char*	Candidate's name, public
votes_	int	Number of Votes, public
party_	char	Name of the party, public
party_rank_	int	The rank of the candidate within a party, public

Party_Data:

Attribute Name	Type	Description
name_	char*	Party's name, public
votes_	int	Number of Votes, public
candidates_	Candidate_Data*	An array of Candidate_Data, public
seats_	int	Number of Seats, public
num_candidates_	int	Number of Candidates, public

File_Data:

Attribute Name	Type/Return Type	Description
voting_type_	char*	Voting type, either CPL or OPL, private
num_parties_	int	Number of Parties, private
party_list_	Party_Data*	Array of Party_Data, private
num_seats_	int	Number of Seats, private
num_ballots_	int	Number of ballots, private
num_candidates_	int	Number of candidates, private
candidates_	Candidate_Data*	Array of Candidate's data, private
quota_	float	Quota number, private
File_Data()		Default constructor of File_Data, public

File_Data(char*)		Constructor, public
Load_File(char*)	void	Load file method, private
Convert_OPL_To_CPL()	void	Convert OPL format to CPL format, private
Generate_Party_Data()	void	Generate Party_Data field, returns void, private
Check_File()	void	Check correctness of file format, private
Break_Tie()	void	Processing tie, public
IsTie()	bool	Returns true if there is a tie, private
get_party_list_()	Party_Data*	Get party_Data structure, public
get_voting_type_()	char*	Get voting type, public
get_num_parties_()	int	Get number of parties, public
get_num_seats_()	int	Get number of seats, public
get_num_ballots_()	int	Get number of ballots, public
get_num_candidates_()	int	Get number of candidates, public
get_candiadtes_()	Candidate_Data*	Get Candidate_Data structure, public
get_votes_totals_()	int*	Get number of votes array, public
get_quota_()	float	Get quota, public

Voting_System

Attribute Name	Type/Return Type	Description
data_	File_Data	File_Data, private
elected_	Candidate_Data*	Candidate_Data that's been elected, private
Voting_System()		Default constructor, public
Voting_System(char*)		Constructor, public
Audit_File()	void	Generates audit file, public
Media_File()	void	Generates media file, public
Display_Winner()	void	Display winner(s) on console, public

5. COMPONENT DESIGN

System Sequence Diagram

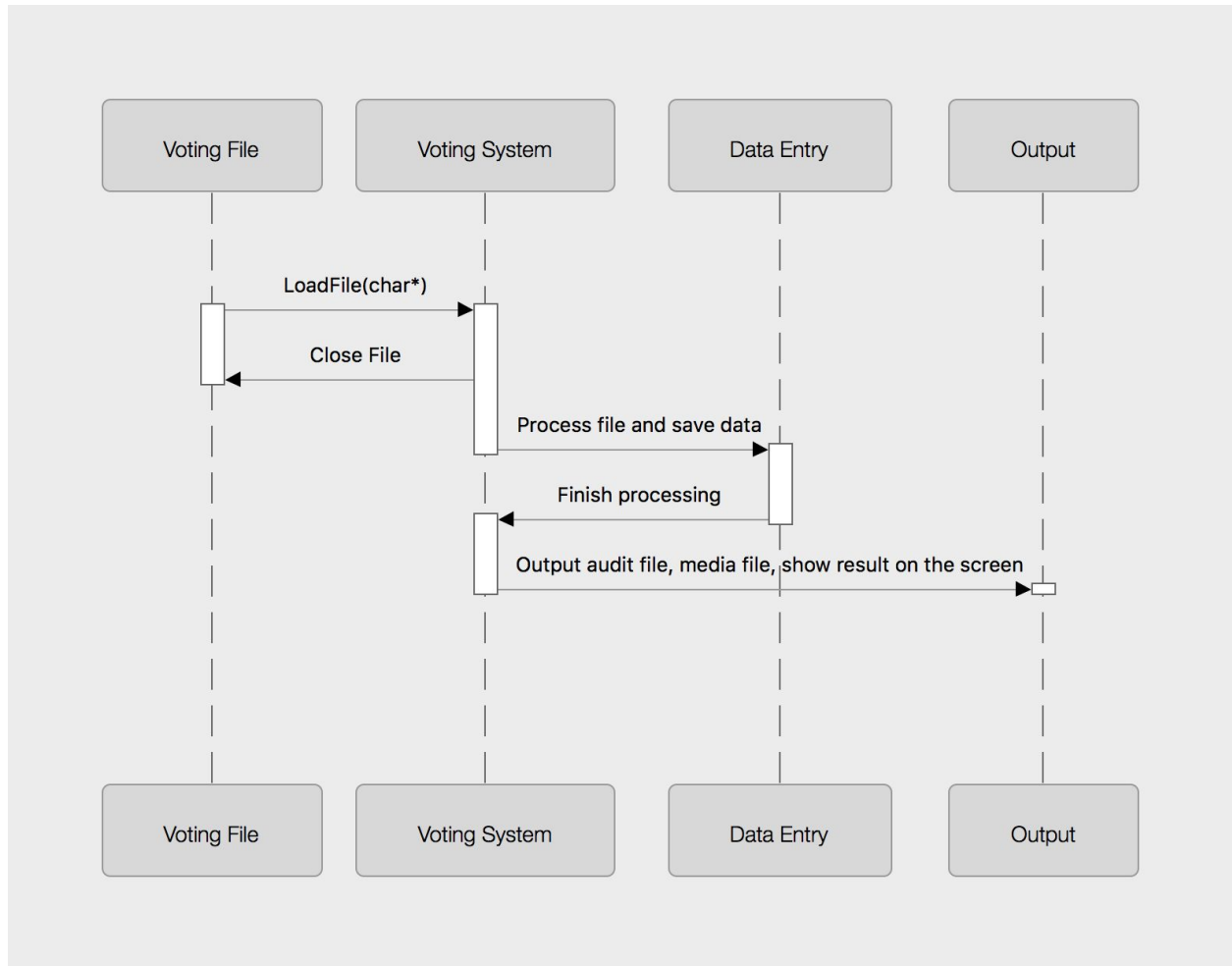


Figure 5.1: System Sequence Diagram

Use Case realizations

Use Case: Tie breaking

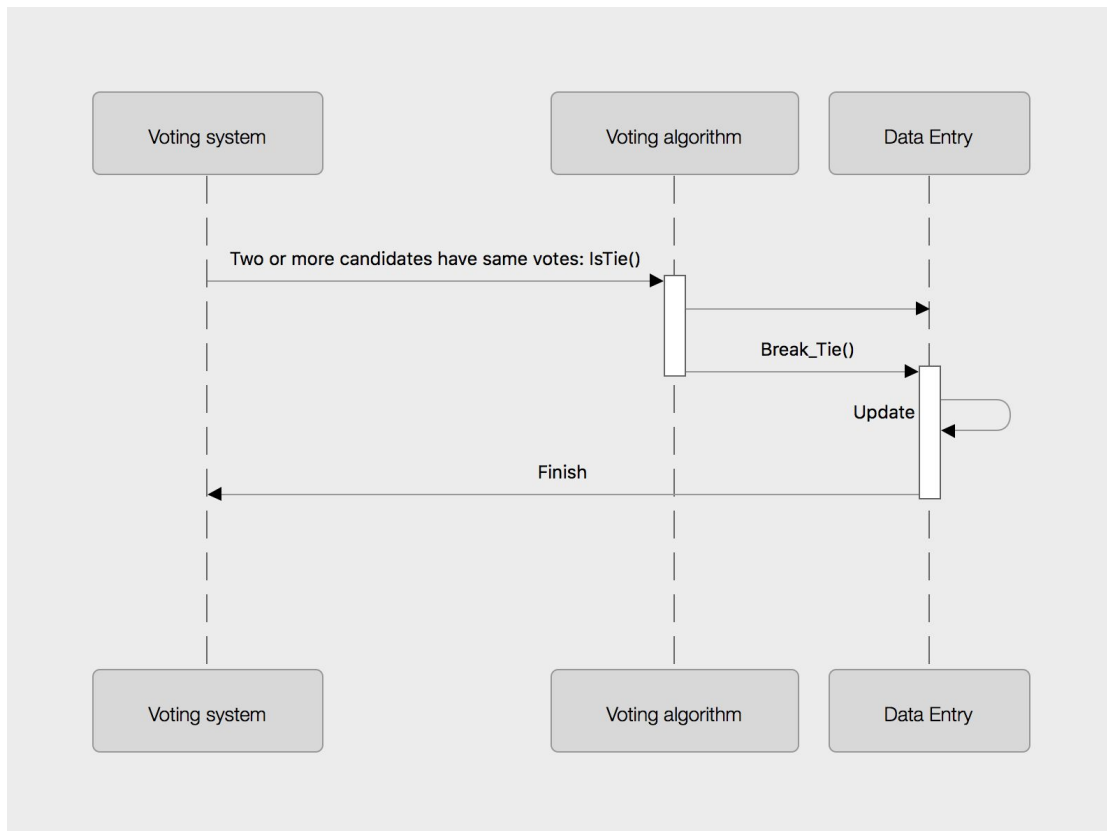


Figure 5.2: Sequence Diagram for Tie Breaking

Use Case: File validation

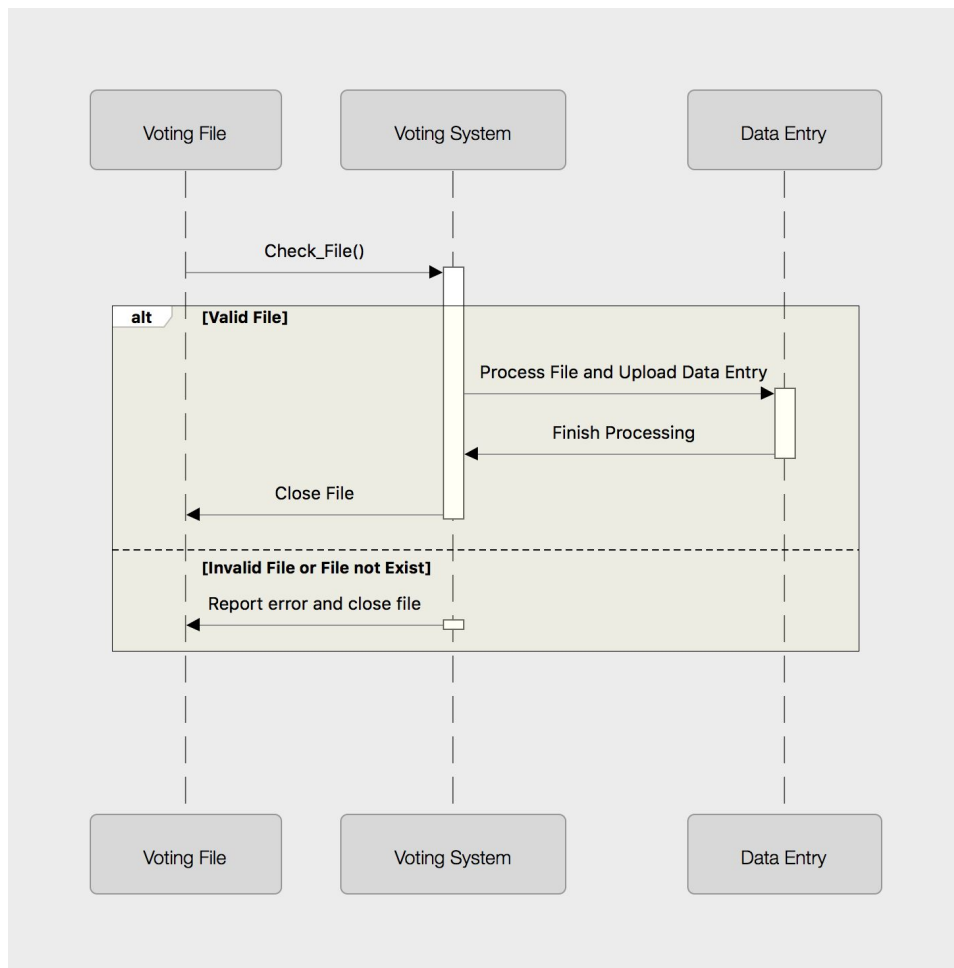


Figure 5.3: Sequence Diagram for File Validation

Use Case: Run OPL/CPL tabulation

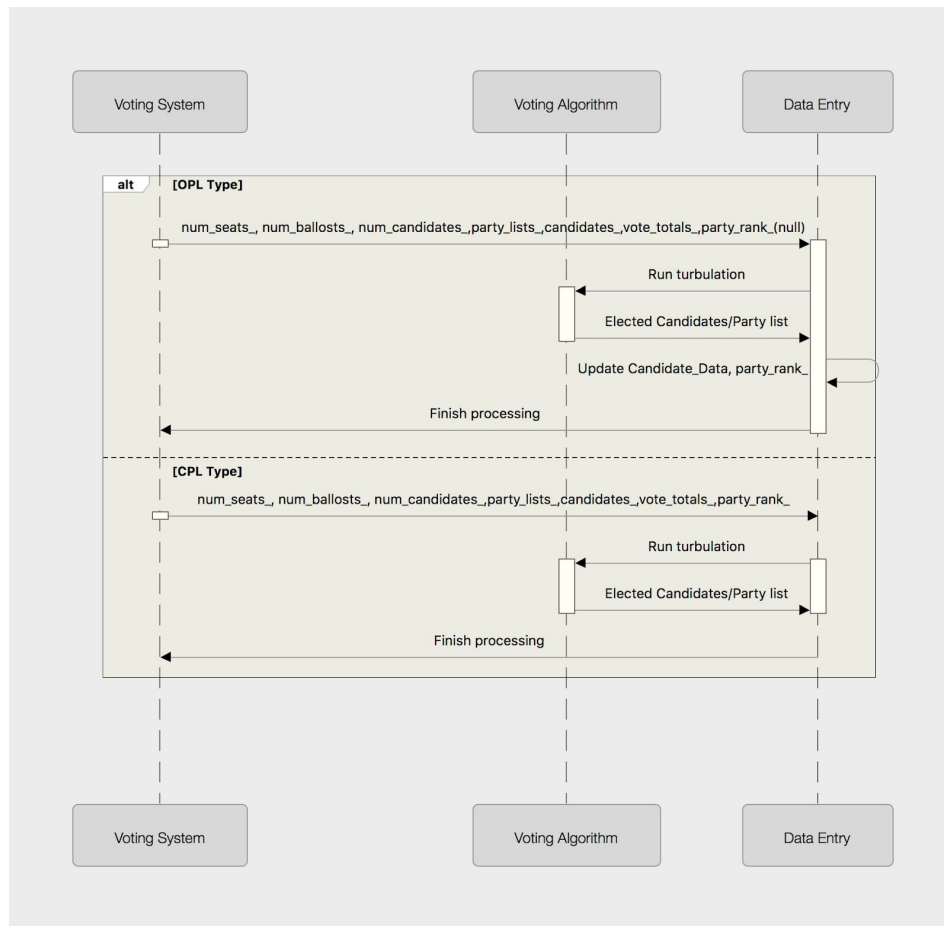


Figure 5.4: Sequence Diagram for OPL/CPL Tabulation

Use Case: Produce Audit File

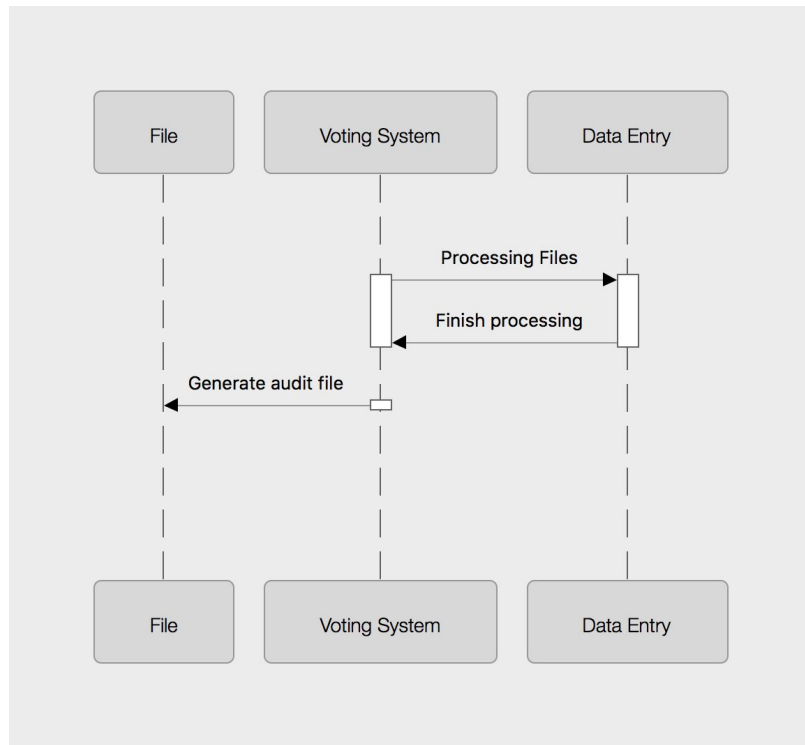


Figure 5.5: Sequence Diagram for Producing Audit File

Use Case: Produce Media File

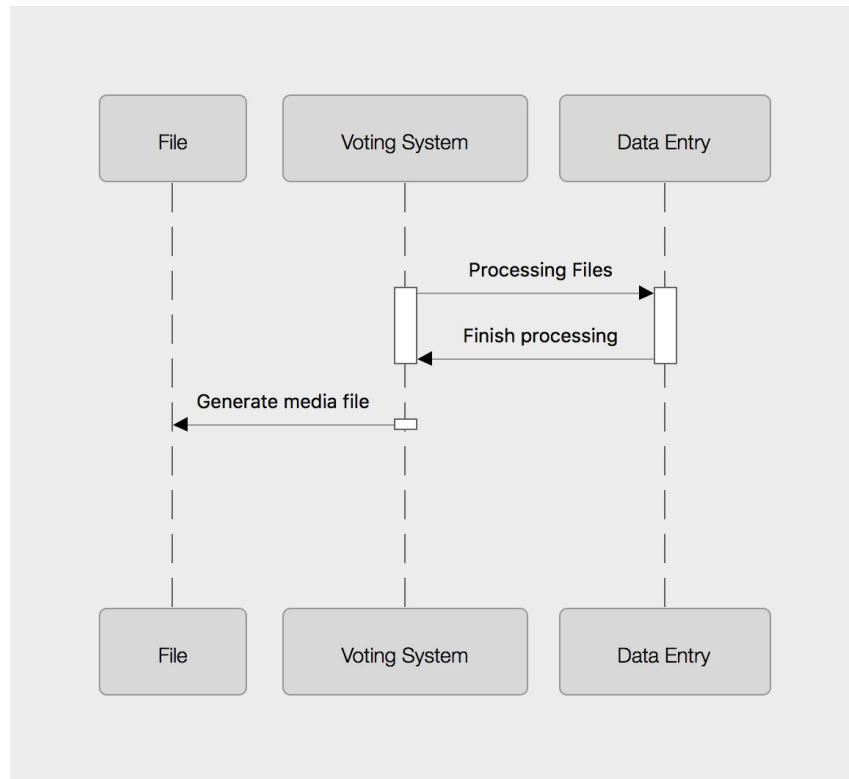


Figure 5.6: Sequence Diagram for Producing Media File

Use Case: Show Winner

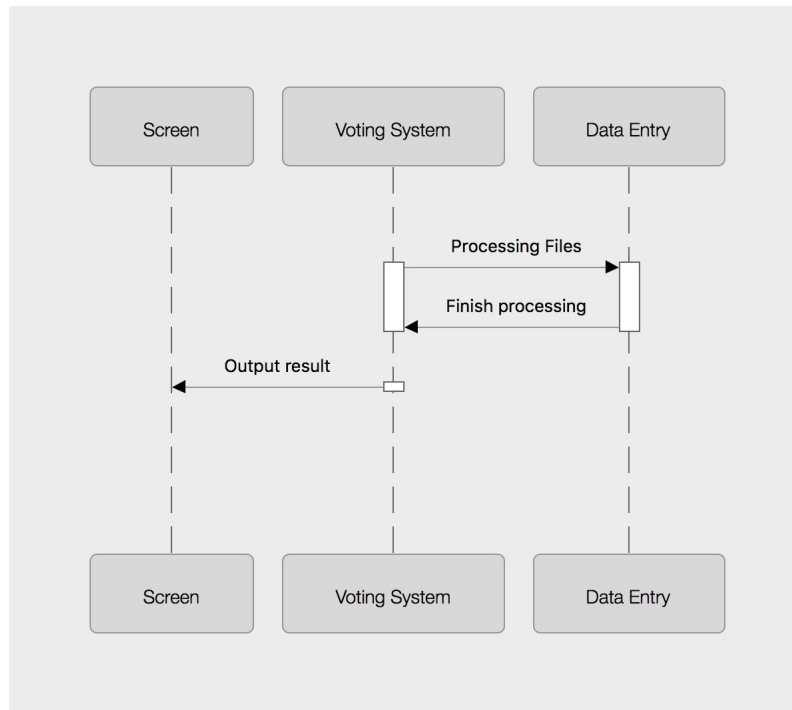


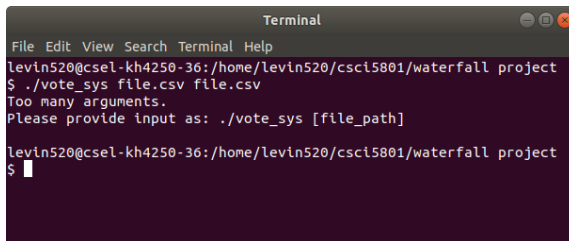
Figure 5.7: Sequence Diagram for Showing Winner

6. HUMAN INTERFACE DESIGN

6.1 Overview of User Interface

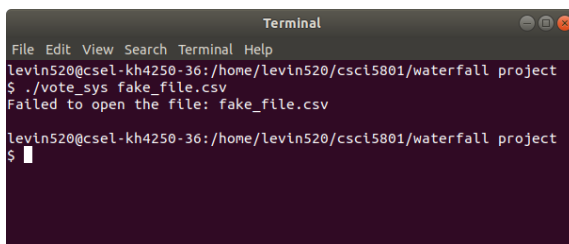
The 5801 Voting System is designed to run through the UNIX command terminal. The user will run the program by executing `./vote_sys [file_path]`. When execution with a valid election file, the program will tabulate and write the results of the election to the display (as shown in figure 6.3). In addition, on successful execution, the program will create a new directory uniquely named based on the given file and the current date, placing inside the audit and media files for the election results. When given incorrect execution parameters, the program will write a warning to the display describing the issue (as shown in figures 6.1, 6.2, 6.4, and 6.5).

6.2 Screen Images



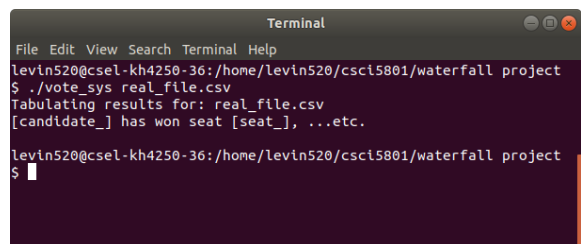
```
Terminal
File Edit View Search Terminal Help
levin520@cse1-kh4250-36:/home/levin520/csci5801/waterfall project
$ ./vote_sys file.csv file.csv
Too many arguments.
Please provide input as: ./vote_sys [file_path]
levin520@cse1-kh4250-36:/home/levin520/csci5801/waterfall project
$
```

Figure 6.1: Voting System executed with too many arguments.



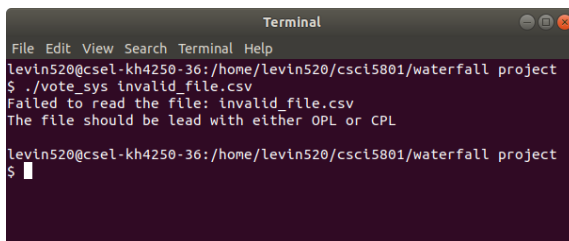
```
Terminal
File Edit View Search Terminal Help
levin520@cse1-kh4250-36:/home/levin520/csci5801/waterfall project
$ ./vote_sys fake_file.csv
Failed to open the file: fake_file.csv
levin520@cse1-kh4250-36:/home/levin520/csci5801/waterfall project
$
```

Figure 6.2: Voting System executed with non-existent file.



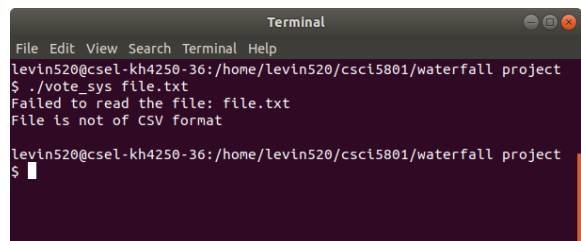
```
Terminal
File Edit View Search Terminal Help
levin520@cse1-kh4250-36:/home/levin520/csci5801/waterfall project
$ ./vote_sys real_file.csv
Tabulating results for: real_file.csv
[candidate_] has won seat [seat_], ...etc.
levin520@cse1-kh4250-36:/home/levin520/csci5801/waterfall project
$
```

Figure 6.3: Voting System executed with valid file.



```
Terminal
File Edit View Search Terminal Help
levin520@cse1-kh4250-36:/home/levin520/csci5801/waterfall project
$ ./vote_sys invalid_file.csv
Failed to read the file: invalid_file.csv
The file should be lead with either OPL or CPL
levin520@cse1-kh4250-36:/home/levin520/csci5801/waterfall project
$
```

Figure 6.4: Voting System executed with improperly formatted file.



```
Terminal
File Edit View Search Terminal Help
levin520@cse1-kh4250-36:/home/levin520/csci5801/waterfall project
$ ./vote_sys file.txt
Failed to read the file: file.txt
File is not of CSV format
levin520@cse1-kh4250-36:/home/levin520/csci5801/waterfall project
$
```

Figure 6.5: Voting System executed with with improper file format.

6.3 Screen Objects and Actions

Figure 6.1 shows the resulting display when the program is executed with too many arguments. A warning will be written to the display, along with the expected execution parameters.

Figure 6.2 shows the resulting display when the program is executed on a non-existent file or given an incorrect file path. When this occurs, a warning that the file could not be opened is written to the display.

Figure 6.3 shows the results of executing the program on a valid file. Here, the program alerts the user that the results are being tabulated, once the calculations are completed, the winning candidates and their elected seats will be written to the display.

Figures 6.4 and 6.5 show the resulting output given incorrect file formats. When this occurs, a warning is written to the display, alerting the user to the issue with their given input file.

7. REQUIREMENTS MATRIX

Component	Data Structure	Use Case	Description
Voting_System	-	VS01	File loaded in
	-	VS04	File validation
	-	VS02	Ballot read in
File_Data	Party_Data Candidate_Data	VS06	CPL ballot tabulation
		VS05	OPL ballot tabulation
		VS08	Determine winner in CPL
		VS07	Determine winner in OPL
		VS03	Tie breaking
Voting_System	-	VS10	Produce audit for CPL
	-	VS09	Produce audit for OPL
	-	VS11	Produce media file
	-	VS12	Show winner