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 1016­-1998](http://www.cs.concordia.ca/%7Eormandj/comp354/2003/Project/ieee-SDD.pdf)[[1]](#footnote-0) for the full IEEE Recommended Practice for Software Design Descriptions.

Team 3

**Voting System**

Software Design Document

Names:

Hailin Archer - deak0007

Bryan Baker - bake1358

Colin Kluegel - klue0037

Josh Spitzer-Resnick - spitz123

Class: CSCI 5801

Semester: Spring 2020

Date: (03/20/2020)

**TABLE OF CONTENTS**

[**1. INTRODUCTION**](#_heading=h.i8xywdq1z2j5) **2**

[1.1 Purpose](#_heading=h.mr4x8iq2v1zs) 2

[1.2 Scope](#_heading=h.wvhcc89do0tg) 2

[1.3 Overview](#_heading=h.razo2ge83a2s) 2

[1.4 Reference Material](#_heading=h.63c3ili9n86f) 2

[1.5 Definitions and Acronyms](#_heading=h.vy936vu13y5i) 2

[**2. SYSTEM OVERVIEW**](#_heading=h.houcqza1d72w) **3**

[**3. SYSTEM ARCHITECTURE**](#_heading=h.lbxc14vgrmo8) **4**

[3.1 Architectural Design](#_heading=h.b8obowzh707l) 4

[3.2 Decomposition Description](#_heading=h.8vbrf1z9kzsw) 5

[3.3 Design Rationale](#_heading=h.mehutlwou0o2) 5

[**4. DATA DESIGN**](#_heading=h.40jl9u8y5wjw) **6**

[4.1 Data Description](#_heading=h.vn8f2bu6kdoh) 6

[4.2 Data Dictionary](#_heading=h.wd4wm36waxlj) 6

[**5. COMPONENT DESIGN**](#_heading=h.21ixsy4vlzpx) **6**

[**6. HUMAN INTERFACE DESIGN**](#_heading=h.urhjepoexc8) **7**

[6.1 Overview of User Interface](#_heading=h.107c37n6zbbo) 7

[6.2 Screen Images](#_heading=h.rmunt9q6076u) 8

[6.3 Screen Objects and Actions](#_heading=h.w57zqsacz8cg) 10

[**7. REQUIREMENTS MATRIX**](#_heading=h.wlj6xyxxhi8k) **11**

[**8. APPENDICES**](#_heading=h.ykxehd2awlp0) **11**

## 1. INTRODUCTION

### 1.1 Purpose

*Identify the purpose of this SDD and its intended audience. (e.g. “This software design document describes the architecture and system design of XX. ….”).*

This Software Design Document provides the design details of the Voting system.

The expected audience are our class instructor, TA and our team members.

### 1.2 Scope

*Provide a description and scope of the software and explain the goals, objectives and benefits of your project. This will provide the basis for the brief description of your product.*

This document contains a complete description of the design of the Voting System.

The goal of the voting system is to provide users with election results of STV (Single Transferable Vote) algorithm and plurality algorithm. Users will need to provide the system with the number of seats, select election algorithm and csv files containing ballot information.

The basic architecture is a stand alone program consisting of following major objects: UserInterface, Election, ElectionRecord, ResultDisplay, Candidate, Ballot, Logger.

### 1.3 Overview

*Provide an overview of this document and its organization.*

The remaining chapters and their contents are listed below.

Section 2 is system overview.

Section 3 is the Architectural Design that specifies the objects that collaborate to perform all the functions included in the system. Each of these objects has an Abstract Description concerning the services that it provides to the rest of the system.

Section 4 lists Data Structure Design.

Section 5 describes Components.

Section 6 discusses the User Interface Design.

Section 7 shows the relationship between the VS system’s components and SRS requirements.

### 1.4 Reference Material

*This section is optional.*

*List any documents, if any, which were used as sources of information for the test plan.*

https://www.slideshare.net/peny\_mg/sdd-software-des-sample

### 1.5 Definitions and Acronyms

*This section is optional.*

Provide definitions of all terms, acronyms, and abbreviations that might exist to properly interpret the SDD. These definitions should be items used in the SDD that are most likely not known to the audience.

|  |  |
| --- | --- |
| Term | Definition |
| VS | Voting System |
| STV | Single Transferable Vote |
| Plurality |  |
| SRS | System Requirements Specification |

## 2. SYSTEM OVERVIEW

Give a general description of the functionality, context and design of your project. Provide any background information if necessary.

The software system being developed is a voting system to be used in local elections. The system will be designed to automate the counting of ballots to simplify the running of elections. The main feature of the software will be to run two types of elections, a plurality voting election and a single transferable voting (STV) election.

In addition to its primary purpose of running an election the software will need to provide some additional features. The software needs to display detailed information about the election results, that is, it should display the number of ballots, the number of seats, the number of candidates and the winner(s) of the election. The software will also need to create a detailed report that will act as an audit for the election. The report will be saved as a text file and show details about how ballots were assigned to candidates as the election progressed.

The software will also require a diagnostic mode. The diagnostic mode will be entered using a command line option. The diagnostic mode is required to support an option to disable ballot shuffling so the system can be calibrated. The diagnostic mode will also have options for developers to use to debug the software.

To aid the user of the voting system a help window will be provided that will give the user information about how to run the program.

The above information leads to a list of the following sub-systems within the voting system:

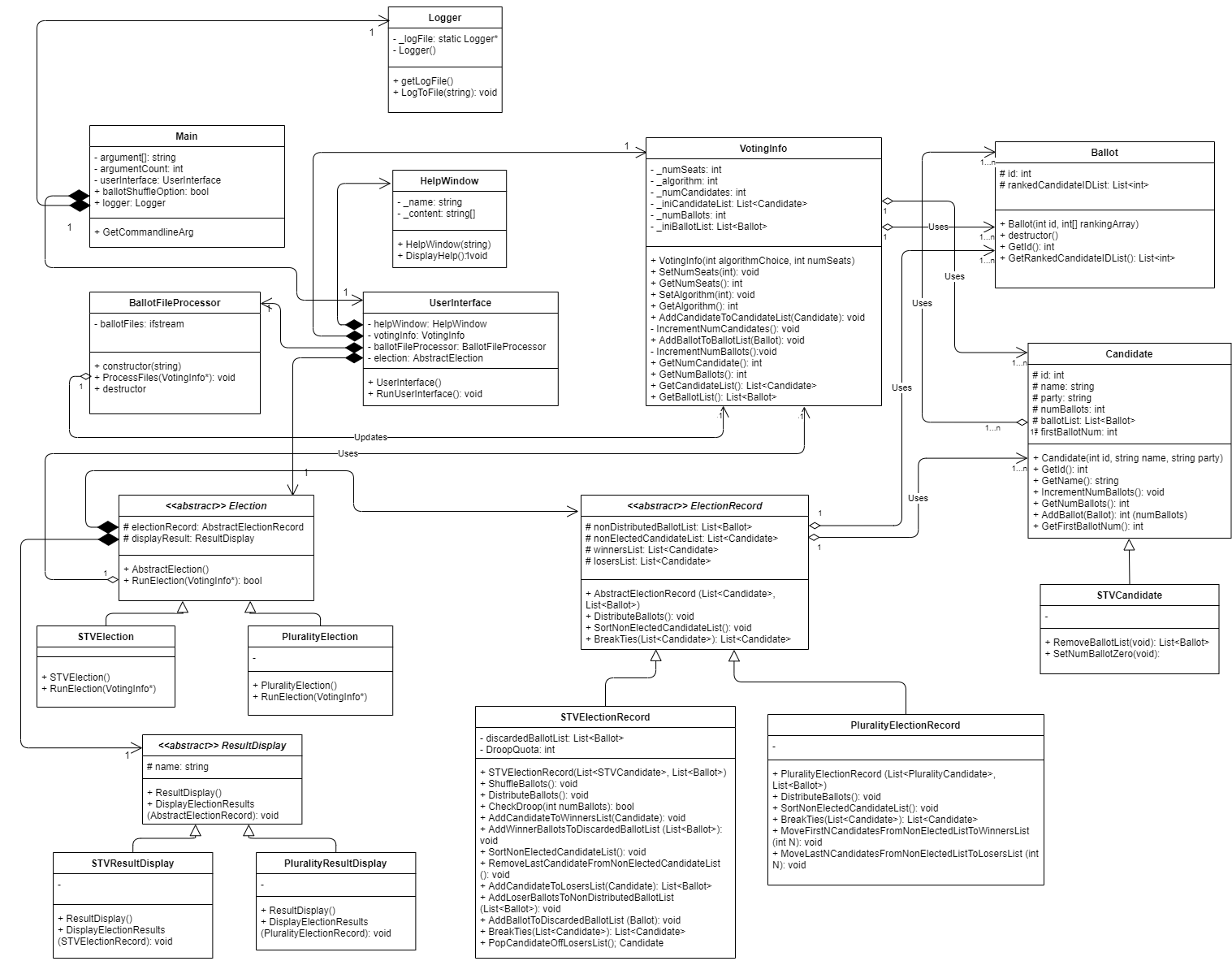
* Ballot handling system
* Plurality voting system
* Single transferable voting (STV) system
* Audit reporting system
* Diagnostic system
* Help / display system

## 3. SYSTEM ARCHITECTURE

### 3.1 Architectural Design

*Develop a modular program structure and explain the relationships between the modules to achieve the complete functionality of the system. This is a high level overview of how responsibilities of the system were partitioned and then assigned to subsystems. Identify each high level subsystem and the roles or responsibilities assigned to it. Describe how these subsystems collaborate with each other in order to achieve the desired functionality. Don’t go into too much detail about the individual subsystems. The main purpose is to gain a general understanding of how and why the system was decomposed, and how the individual parts work together. Provide a diagram showing the major subsystems and data repositories and their interconnections. Describe the diagram if required.*

Will insert the UML diagram here and make a detailed description of each class, like the SDD example.



**Main**

Name: Main

Type: Class

Description: This is the class the operating system calls upon program execution. Main class will define a global variable ballotShuffleOption and initialize it to true or false based on the command line argument in the main constructor. Main class will also instantiate the Logger class. Then Main class will instantiate the UserInterface class and call its RunUserInterface method.

Data members:

* \_userInterface:

Type: UserInterface class

Access: Private

* ballotShuffleOption:

Type: boolean

Access: public

* logger:

Type: Logger class

Access: public

Constructors:

* main(argc int, argv\*\* char): int

Arguments: argc - argument count, argv - arguments strings

Returns: int (0 - no error, others - error)

Access: public

* main(): int

Arguments: none

Returns: int (0 - no error, others - error)

Access: public

Member Functions: none

Flow of Events:

1. Program starts running
2. Define public static variable ballotShuffleOption and initialize it to False
3. Check command line arguments. If there is the command line argument ‘-t’, set ballotShuffleOption to True
4. Instantiate logger
5. Instantiate userInterface
6. Call userInterface.RunUserInterface
7. Return 0

**Logger**

Name: Logger

Type: Class

Description: Logger class is instantiated in main and accessible by all objects of the program.

Data Members:

* \_logFile: static Logger\*

Type: static Logger\*

Access: private

Constructor:

* Logger(): void

Arguments: none

Returns: none

Access: private

Member Functions:

* GetLogFile(): string

Arguments: none

Returns: string

Access: public

Description: Return value of data member \_logFile

* LogToFile(string): void

Arguments: string

Returns: none

Access: public

Description: Write input string in \_logFile

**UserInterface**

Name: UserInterface

Type: Class

Description: This form will launch automatically after the program starts running. The form will have a combination of list boxes and blank fields to be completed. Some fields will be marked as required. There will also be a help option. When a user chooses the help option he/she will be presented with information on how to run the election. When the form is completed the user can click on the run election button.

Data Members:

* helpWindow: HelpWindow

Type: HelpWindow Class

Access: Private

* votingInfo: VotingInfo

Type: VotingInfo Class

Access: Private

* ballotFileProcessor: BallotFileProcessor

Type: BallotFileProcessor Class

Access: Private

* election: AbstractElection

Type: AbstractElection Class

Access: Private

Constructors:

* UserInterface()

Arguments: none

Returns: none

Access: public

Description: Instantiate UserInterface Class

Member Functions:

* RunUserInterface(): void

Arguments: none

Returns: none

Access: public

Description: Launch the user interface

Flow of Events:

1. Declare data members
2. User is presented with the form
3. If user chooses help option
   1. Run helpWindown.DisplayHelp();
4. User fills in boxes and selects from list boxes
   1. Check validity of input values
   2. Instantiate VotingInfo object with values from the form
5. User clicks run election button
   1. Check if VotingInfo is properly instantiated
6. If the election algorithm is STV, instantiate election to STVElection Concrete Class. If the election algorithm is Plurality, instantiate election to PluralityElection Concrete Class
7. Call ballotFileProcessor.ProcessFiles(votingInfo)
8. Call election.RunElection()

**HelpWindow**

Name: HelpWindow

Type: Class

Description: This window launches when the user chooses the help option from the user interface. This window displays strings (information on how to use VS to run elections). This window does not have a form for the user to input information. It has an option for the user to return to the user interface.

Data Members: none

Constructors/Destructor:

* HelpWindow()

Arguments: none

Returns: none

Access: public

* ~HelpWindow()

Member Functions:

* DisplayHelp(): void

Arguments: none

Returns: none

Access: public

Description: This function displays the window and help content

Flow of Events:

1. User views help contents
2. User closes the window
3. Program returns to UserInterface

**BallotFileProcessor**

Name: BallotFileProcessor

Type: Class

Description: This class is instantiated once in UserInterface class. It reads ballot file(s) line by line. It extracts candidate information first and creates a candidate list in the VotingInfo object. It then reads ballot information one at a time and updates the ballot list in the VotingInfoObject.

Data Members:

* ballotFiles

Type: ifstream

Access: private

Constructors/Destructor:

* BallotFileProcessor(string)

Arguments: string fileName - Ballot File Name to be processed

Returns: none

Access: public

* ~BallotFileProcessor()

Member Functions:

* ProcessFiles(VotingInfo\*): void

Arguments: VotingInfo\* votingInfo

Returns: none

Access: public

Description: This function opens Ballot files. It reads in content line by line. The function extracts the candidates’ information from the header of the first file, constructs Candidate objects, then updates the candidate list in the VotingInfo object. The function then reads ballots line by line, constructs Ballot objects, then updates the ballot list in the VotingInfo object.

**VotingInfo**

Name: VotingInfo

Type: Class

Description: VotingInfo class stores information needed for election. It is instantiated once in the UserInterface object. It is passed as a reference argument to BallotFileProcessor object method and Election object method.

Data Members:

* \_numSeats

Type: int

Access: private

* \_algorithm

Type: int

Access: private

* \_numCandidates

Type: int

Access: private

* \_iniCandidateList

Type: List<Candidate>

Access: private

* \_numBallots

Type: int

Access: private

* \_iniBallotList

Type: List<Ballot>

Access: private

Constructors/Destructor:

* VotingInfo (int algorithmChoice, int numSeats)

Arguments: int algorithmChoice, int numSeats

Returns: none

Access: public

* ~VotingInfo()

Member Functions:

* GetNumSeats(): int

Arguments: none

Returns: int

Access: public

Description: Get value of private member \_numSeats

* GetAlgorithm(): int

Arguments: none

Returns: int

Access: public

Description: Get value of private member \_algorithm

* AddCandidateToCandidateList(Candidate): void

Arguments: Candidate - Candidate object

Returns: none

Access: public

Description: Add a candidate object to private member \_iniCandidateList

* IncrementNumCandidates(): void

Arguments: none

Returns: none

Access: private

Description: This function is called by AddCandidateToCandidateList method. It increments private member \_numCandidates when a Candidate object is added to \_iniCandidateList

* AddBallotToBallotList(Ballot): void

Arguments: Ballot - a Ballot object

Returns: none

Access: public

Description: Add a ballot object to private member \_iniBallotList

* IncrementNumBallots():void

Arguments: none

Returns: none

Access: public

Description: This function is called by AddBallotToBallotList method. It increments private member \_numBallots when a Ballot object is added to \_iniBallotList

* GetNumCandidate(): int

Arguments: none

Returns: int

Access: public

Description: Returns value of private member \_numCandidate

* GetNumBallots(): int

Arguments: none

Returns: int

Access: public

Description: Returns value of private member \_numBallot

* GetCandidateList(): List<Candidate>

Arguments: none

Returns: List<Candidate>

Access: public

Description: Returns values of private member \_iniCandidateList

* GetBallotList(): List<Ballot>

Arguments: none

Returns: List<Ballot>

Access: public

Description: Returns values of private member \_iniBallotList

**Ballot**

Name: Ballot

Type: Class

Description: An object defined by this class stores information from a single ballot. Its data member \_id is unique to each Ballot object, assigned sequentially when BallotFileProcessor generates Ballot objects. Its data member \_rankedCandidateIDList stores ranked Candidate IDs based on the ranking from original ballots (i.e. if a ballot ranked candidates A,B,C,D,E: 0,1,0,2,3, the \_rankedCandidateIDList will be {2,4,5}) .

Data Members:

* \_id

Type: int

Access: private

* \_rankedCandidateIDList

Type: List<int>

Access: private

Constructors/Destructor:

* Ballot(int id, int[] rankingArray)

Arguments: int id, int[] rankingArray

Returns: none

Access: public

Description: This constructor takes an integer and the raw ranking array from ballots, assign integer input to private data member \_id, convert the raw ranking array to the rankedCandidateIDList and store it in the corresponding data member

* ~Ballot()

Member Functions:

* GetId(): int

Arguments: none

Returns: int

Access: public

Description: Returns value of private data member \_id

* GetRankedCandidateIDList(): int

Arguments: none

Returns: List<int>

Access: public

Description: Returns value of private data member \_rankedCandidateIDList

**Candidate**

Name: Candidate

Type: Class

Description: An object defined by this class stores information of a candidate from the original ballot files and associated methods. Its data member \_id is a unique integer, assigned sequentially when BallotFileProcessor generates each object. Its data member \_ballotList stores ballot objects assigned to the candidate object.

Data Members:

* \_id

Type: int

Access:



Type:

Access:



Constructors:



Arguments:

Returns:

Access:



Member Functions:



Arguments:

Returns:

Access:

Description:

**STVCandidate**

Name:

Type:

Description:

Data Members:



Type:

Access:



Type:

Access:



Constructors:



Arguments:

Returns:

Access:



Member Functions:



Arguments:

Returns:

Access:

Description:

**Election**

Name:

Type:

Description:

Data Members:



Type:

Access:



Type:

Access:



Constructors:



Arguments:

Returns:

Access:



Member Functions:



Arguments:

Returns:

Access:

Description:

**STVElection**

Name:

Type:

Description:

Data Members:



Type:

Access:



Type:

Access:



Constructors:



Arguments:

Returns:

Access:



Member Functions:



Arguments:

Returns:

Access:

Description:

**PluralityElection**

Name:

Type:

Description:

Data Members:



Type:

Access:



Type:

Access:



Constructors:



Arguments:

Returns:

Access:



Member Functions:



Arguments:

Returns:

Access:

Description:

**ElectionRecord**

Name:

Type:

Description:

Data Members:



Type:

Access:



Type:

Access:



Constructors:



Arguments:

Returns:

Access:



Member Functions:



Arguments:

Returns:

Access:

Description:

**STVElectionRecord**

Name:

Type:

Description:

Data Members:



Type:

Access:



Type:

Access:



Constructors:



Arguments:

Returns:

Access:



Member Functions:



Arguments:

Returns:

Access:

Description:

**PluralityElectionRecord**

Name:

Type:

Description:

Data Members:



Type:

Access:



Type:

Access:



Constructors:



Arguments:

Returns:

Access:



Member Functions:



Arguments:

Returns:

Access:

Description:

**ResultDisplay**

Name:

Type:

Description:

Data Members:



Type:

Access:



Type:

Access:



Constructors:



Arguments:

Returns:

Access:



Member Functions:



Arguments:

Returns:

Access:

Description:

**STVResultDisplay**

Name:

Type:

Description:

Data Members:



Type:

Access:



Type:

Access:



Constructors:



Arguments:

Returns:

Access:



Member Functions:



Arguments:

Returns:

Access:

Description:

**PluralityResultDisplay**

Name:

Type:

Description:

Data Members:



Type:

Access:



Type:

Access:



Constructors:



Arguments:

Returns:

Access:



Member Functions:



Arguments:

Returns:

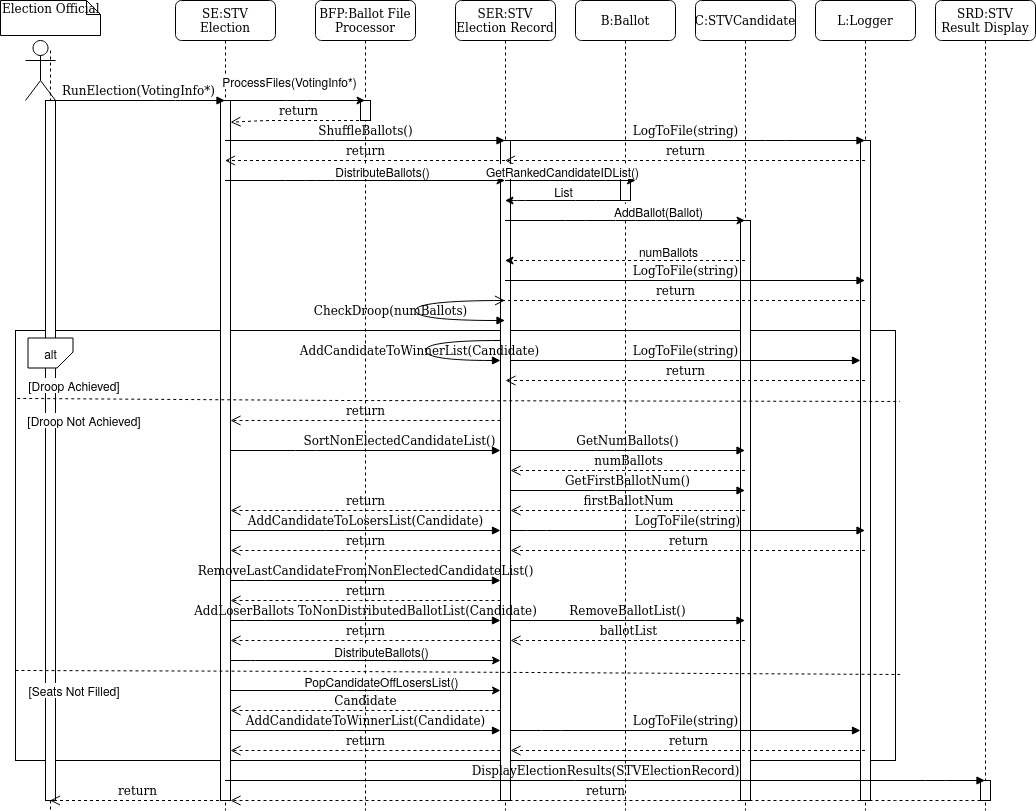
Access:

Description:

### 3.2 Decomposition Description

Provide a decomposition of the subsystems in the architectural design. Supplement with text as needed. You may choose to give a functional description or an object­ oriented description. For a functional description, put top ­level data flow diagram (DFD) and structural decomposition diagrams. For an OO description, put subsystem model, object diagrams, generalization hierarchy diagram(s) (if any), aggregation hierarchy diagram(s) (if any), interface specifications, and sequence diagrams here.

The sequence diagram for the SVT system is given below.



To run the STV election the user first calls the RunElection method from the STV Election object. This method runs the STV election to completion. First the ballots are read in using the Ballot File Processor. This processor creates a ballot object for each line of the ballot file. It also processes the ballots to create a sorted list in the order of prefered candidates for that particular ballot. The ballots are then shuffled using the STV Election Record object and the shuffle is recorded in the Logger. The STV Election object then calls the DistributeBallot method to start issuing ballots to candidates. This method loops through all of the ballot objects and for each ballot calls the GetRankedCandidateIDList method to get the list of candidates for that ballot in order of voter preference. The STV Election Record then determines the prefered candidate by comparing the voter preference to the list of candidates that have not won. The preferred candidate for that ballot is determined and then the addBallot method for the STVCandidate object is called to add that ballot to that candidates ballot list. The number of ballots for that candidate is returned and the data is recorded in the Logger. Droop is then checked for this candidate.

If droop is achieved then the winner is added to the winner list and the data is recorded in the Logger.

If droop is not achieved and all of the ballots have been distributed then the process returns to the STV Election method. This method calls the SortNonElectedCandidateList method in the STV Election Record object which gets the number of ballots and the first ballot id for each candidate on that list to determine the loser. That candidate is added to the losers list and the data is captured in the Logger.

The losing candidates ballots are then moved back to the NonDistributedBallotList and the DistributeBallot method is called again to distribute the ballots in the list.

Once the NonElectedCandidatesList is empty, if the seats are still not full then PopCandidateOffLosersList method from the STV Election Record object is called to get the last loser. This candidate is then added to the winners list and the data is recorded in the Logger.

Once all seats have been filled the election results are displayed and the program returns back to the user.

### 3.3 Design Rationale

Discuss the rationale for selecting the architecture described in 3.1 including critical issues and trade/offs that were considered. You may discuss other architectures that were considered, provided that you explain why you didn’t choose them.

The rationale is to have classes for all major system components.

## 4. DATA DESIGN

### 4.1 Data Description

*Explain how the information domain of your system is transformed into data structures. Describe how the major data or system entities are stored, processed and organized. List any databases or data storage items.*

The ballots are stored in Ballot objects, which contain lists of ranked Candidate ids and the current Candidate the Ballot is assigned to.

The candidates are stored in Candidate objects, which contain lists of Ballots assigned to that candidate.

### 4.2 Data Dictionary

*Alphabetically list the system entities or major data along with their types and descriptions. If you provided a functional description in Section 3.2, list all the functions and function parameters. If you provided an OO description, list the objects and its attributes, methods and method parameters.*

## 5. COMPONENT DESIGN

In this section, we take a closer look at what each component does in a more systematic way. If you gave a functional description in section 3.2, provide a summary of your algorithm for each function listed in 3.2 in procedural description language (PDL) or pseudocode. If you gave an OO description, summarize each object member function for all the objects listed in 3.2 in PDL or pseudocode. Describe any local data when necessary.

**main:**

int main(argc int, argv\*\* char) {

static bool ballotShuffleOption = False;

if argv[1] == ‘-t’{

ballotShuffleOption = True;}

initialize class Logger;

initialize class userInterface;

userInterface.RunUserInterface();

return 0;

}

**Logger:**

class Logger {

static string \_logFile\*;

class Logger();

public:

string GetLogFile(void) {

return \_logFile\*;

}

void LogToFile(string) {

write string to \_logFile\*;

return;

}

}

**UserInterface:**

class UserInterface {

class HelpWindow {

(see class details below)

}

class VotingInfo {

(see class details below)

}

class BallotFileProcessor {

(see class details below)

}

class AbstractElection {

(see class details below)

}

public:

RunUserInterface(void){

declare variables;

output form;

if help.click == True{

HelpWindow.DisplayHelp();

}

check data boxes for correct format:

initialize VotingInfo class (VotingInfo\*);

runelectionbtn.click == True {

if VotingInfo\* correct {

if electionType == 0 {

ballotFileProcessor.ProcessFiles(VotingInfo\*);

STVElection.RunElection(VotingInfo\*);

} elif electionType == 1 {

ballotFileProcessor.ProcessFiles(VotingInfo\*);

PluralityElection.RunElection(VotingInfo\*);

} else {

print(“Please specify either STV or Plurality”);

}

} else {

print(“Please correct data”);

}

}

}

}

**HelpWindow:**

class HelpWindow {

public:

void DisplayHelp(void) {

print(windowContents);

wait for user to close window;

return;

}

}

**BallotFileProcessor:**

class BallotFileProcessor {

ifstream ballotFiles (ballotFileName);

public:

void ProcessFiles(VotingInfo\*) {

int linecnt = 1

if (ballotFiles.is\_open()) {

while (getline(ballotFiles, line)) {

if linecnt == 1 {

parse line for list of candidates;

for( i = 0, i < length(candidates), i++) {

initialize candidate;

}

update VotingInfo[CandidateList];

} else {

read line;

initialize Ballot;

update VotingInfo[BallotList];

}

}

ballotFiles.close();

} else {

print(“Cannot open file”);

}

}

}

**VotingInfo:**

class VotingInfo {

int \_numSeats;

int \_algorithm;

int \_numCandidates;

char \_iniCandidateList[];

int \_numBallots;

obj \_iniBallotList[];

public:

int GetNumSeats(void) {

return \_numSeats;

}

int GetAlgorithm(void) {

return \_algorithm;

}

void IncrementNumCandidates(void) {

\_numCandidates += 1;

return;

}

void AddCandidateToCandidateList(Candidate) {

\_iniCandidateList = \_iniCandidateList.append(Candidate);

self.IncrementNumCandidates();

return;

}

void IncrementNumBallots(void) {

\_numBallots += 1;

return;

}

void AddBallotToBallotList(Ballot) {

\_iniBallotList = \_iniBallotList.append(Ballot);

self.IncrementNumBallots();

return;

}

int GetNumCandidate(void) {

return \_numCandidate;

}

int GetNumBallots(void) {

return \_numBallot;

}

List<Candidate> GetCandidateList(void) {

return \_iniCandidateList;

}

List<Ballot> GetBallotList(void) {

return \_iniBallotList;

}

}

**Ballot:**

Ballot(int id, int[] rankingArray)

{

this->id = id

for each element in rankingArray

{

this->ranked\_candidateIDList.add(element);

}

}

int GetId()

{

return this->id;

}

List<int> GetRankedCandidateIDList()

{

return this->ranked\_candidateIDList.add(element);

}

## 6. HUMAN INTERFACE DESIGN

### 6.1 Overview of User Interface

Describe the functionality of the system from the user’s perspective. Explain how the user will be able to use your system to complete all the expected features and the feedback information that will be displayed for the user.

The user interface will consist of 3 separate windows, a startup/information gathering window, an election results window, and a help window.

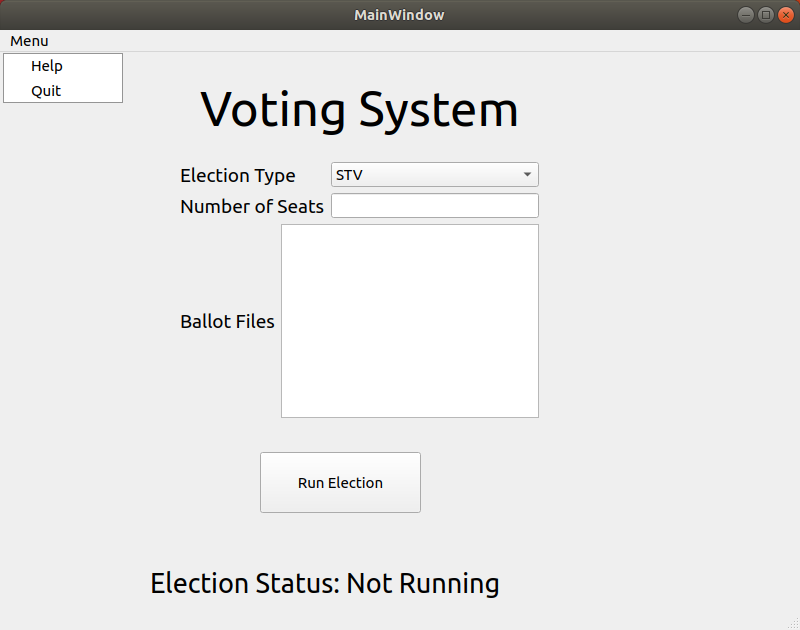
**Startup/Information gathering:** The Startup/information gathering window will be the first window that the user sees when starting the Voting System program. The window will have fields where the user can enter the necessary information to run the election, which includes the election type (STV/plurality), the number of seats, and a field to list the ballot files to be used. The window will also have a button to run election and will show the status of the election (not running/Running/Complete). The window will also have a menu bar where the user can access the help menu.

**Election results window:** The election results window will display information of about the election after it is run. It will display the results of the election and addition information about the election. This additional information will include election type(STV/plurality), number of ballots, number of seats, number of candidates and the droop quota(STV election only). There will also be a menu bar where the user can access the help menu.

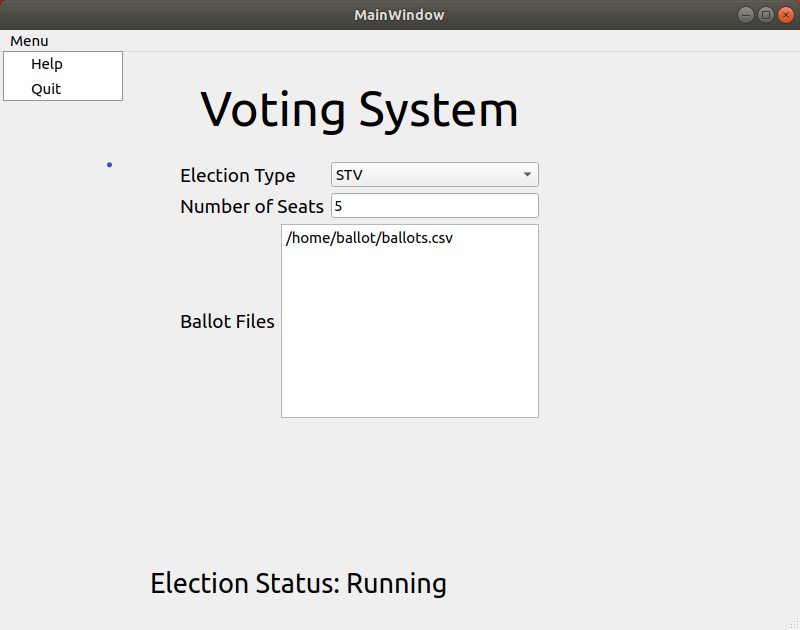
**Help Window:** The help window will display helpful information/ user guide to the user.

### 6.2 Screen Images

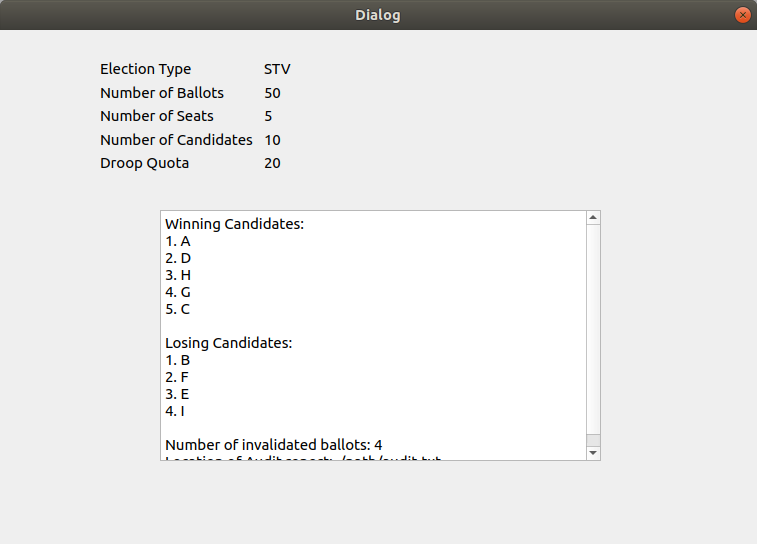
Startup Screen/Election Configuration Screen



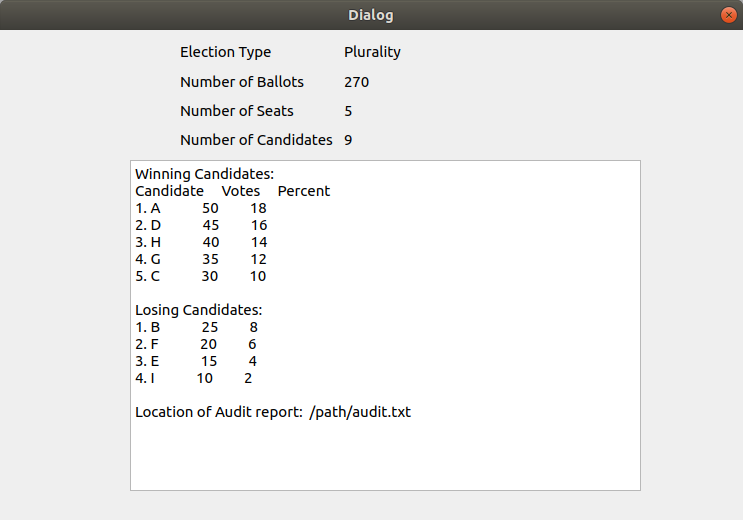
Screen while election is running



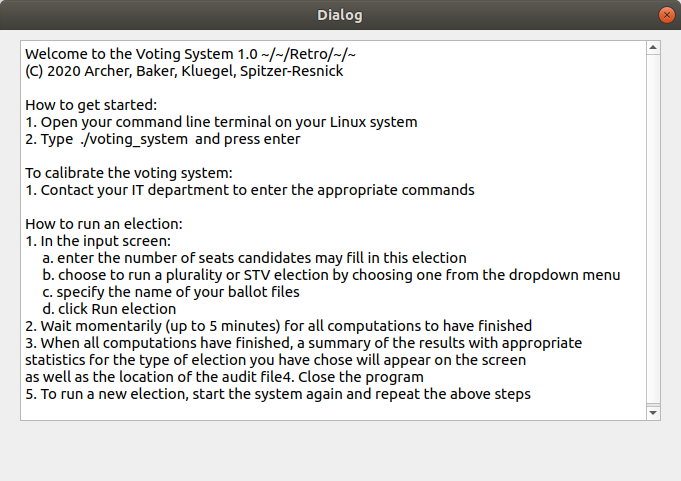
STV election results screen



Plurality election results screen



Help Screen



### 6.3 Screen Objects and Actions

A discussion of screen objects and actions associated with those objects.

**Startup/Information gathering window**: The Startup/information gathering window will have the following screen objects:

* Dropdown menu to allow the user to select the election type, STV or plurality.
* Free form text area where the user enters the number of seats in the election.
* Larger free form text area where the user can list the names of the ballot files to use.
* “Run Election” button that when clicked will start running an election from the user given inputs.
* Election status field that will show when an election is not running(has not been started), is running, or is complete.
* Menu bar that will allow the user to access the help menu.

**Election results window:**  The election election results window will have the following screen objects

* Fields listing information about the election such as Election type, number of ballots, number of seats, number of candidates, and droop quota.
* Non-editable text field that will display the results of the elections.
* Menu bar that will allow the user to access the help menu.

**Help menu:** The help menu will have the following screen objects

* Non-editable text field that will display the help text.
* Menu bar with an exit option to allow the user to exit the window and return to the main program.

## 7. REQUIREMENTS MATRIX

Provide a cross ­reference that traces components and data structures to the requirements in your SRS document.

Use a tabular format to show which system components satisfy each of the functional requirements from the SRS. Refer to the functional requirements by the numbers/codes that you gave them in the SRS.

|  |  |
| --- | --- |
| Use Case | UC\_1 Input required voting information |
| Actors | Election Officials, Ballot handling system |
| Description | An election official enters the information necessary to run the election. This information includes election type(STV or plurality), number of seats, and the ballot files. |
| Data | Election type(STV or plurality), number of seats, list of ballot files. |
| Stimulus | User starts the Voting System program. |
| Response | Voting system is ready to run STV or plurality election. |
| Comments | Number of seats entered should be a positive integer. |

|  |  |
| --- | --- |
| Use Case | UC\_2 Run STV election |
| Actors | Election Officials, Single transferable voting (STV) system |
| Description | All required election information has been entered and an election official now wishes to run an STV election. |
| Data | From the input data Candidate and ballot objects are created to run the election. |
| Stimulus | User clicks ‘Run election’ button. |
| Response | Ballots are processed by the STV election algorithm. |
| Comments | Results will be displayed when algorithm is done running. |

|  |  |
| --- | --- |
| Use Case | UC\_3 Run plurality election |
| Actors | Election Officials, Plurality voting system |
| Description | All required election information has been entered and an election official now wishes to run a plurality election. |
| Data | From the input data Candidate and ballot objects are created to run the election. |
| Stimulus | User clicks ‘Run election’ button. |
| Response | Ballots are processed by the plurality election algorithm. |
| Comments | Results will be displayed when the algorithm is done running. |

|  |  |
| --- | --- |
| Use Case | UC\_4 Run test files(s) |
| Actors | Developers/testers, Diagnostic system |
| Description | A developer/tester may wish to run test files through the voting system either for calibration purposes or to perform unit testing. |
| Data | Election type(STV or plurality), number of seats, list of ballot files (passed in through command line args). The ballot files are used to create candidate and ballot objects to run the election with. |
| Stimulus | User starts the Voting System program with command line arguments. |
| Response | Voting System runs an election based on command line arguments and displays the result. |
| Comments | None. |

|  |  |
| --- | --- |
| Use Case | UC\_5 Turn off ballot shuffle |
| Actors | Developers/testers, Diagnostic system |
| Description | A developer or tester may wish to check that the STV election is being run correctly. By disabling the ballot shuffle the results of the STV election will be predictable. |
| Data | boolean passed in to disable shuffle. |
| Stimulus | User starts the Voting System program with a command line argument. |
| Response | Voting system runs as it normally would but ballots will not be shuffled for the STV election. |
| Comments | Ballot shuffle cannot be turned back on, program needs to be restarted to turn shuffle back on. |

|  |  |
| --- | --- |
| Use Case | UC\_6 Display election results |
| Actors | Election Officials/testers/developers, Plurality voting system or Single transferable voting (STV) system |
| Description | After an election has been run the results of the election will be displayed. The election results are encoded into a String format and printed to the screen. |
| Data | Election type(STV or plurality), number of seats, number of candidates, number of ballots, winners list, losers list, droop quota(STV only). |
| Stimulus | The STV or plurality algorithm has completed. |
| Response | The results are displayed. |
| Comments | None. |

|  |  |
| --- | --- |
| Use Case | UC\_7 Display help window. |
| Actors | Election Officials/developers/testers, Help/display system. |
| Description | A user of the Voting System wants more information about how to run the Voting System. |
| Data | File object loads in text of help file. |
| Stimulus | Users click the ‘help’ button on the menu bar. |
| Response | Help document displayed. |
| Comments | Help window can be displayed while the user is running the election. |

|  |  |
| --- | --- |
| Use Case | UC\_8 Create election audit report |
| Actors | Election Officials/developers/testers, Audit reporting system |
| Description | An audit file needs to be produced for each election. The report shows how the election progressed (how ballots were assigned to candidates). This report is not returned to the screen, it is saved to a text file. |
| Data | File object created to save audit file. |
| Stimulus | When the election is started the audit file will be created. |
| Response | The audit file is updated while the election is running. |
| Comments | None. |

## 8. APPENDICES

*This section is optional.*

Appendices may be included, either directly or by reference, to provide supporting details that could aid in the understanding of the Software Design Document.

1. [http://www.cs.concordia.ca/~ormandj/comp354/2003/Project/ieee](http://www.cs.concordia.ca/%7Eormandj/comp354/2003/Project/ieee)­SDD.pdf [↑](#footnote-ref-0)