

## 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)<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>

CSCI 5801 - Team 5

**Ballot Processor**

Software Design Document

**Names:**

Bek Allenson,  
Perrie Gryniewicz,  
Matthew Johnson, and  
Logan Watters

Date: (10/20/2023)

## TABLE OF CONTENTS

<b>1. 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. SYSTEM OVERVIEW</b>	<b>2</b>
<b>3. SYSTEM ARCHITECTURE</b>	<b>2</b>
3.1 Architectural Design	2
3.2 Decomposition Description	3
3.3 Design Rationale	3
<b>4. DATA DESIGN</b>	<b>3</b>
4.1 Data Description	3
4.2 Data Dictionary	3
<b>5. COMPONENT DESIGN</b>	<b>3</b>
<b>6. 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. REQUIREMENTS MATRIX</b>	<b>4</b>
<b>8. APPENDICES</b>	<b>4</b>

## **1. INTRODUCTION**

### **1.1 Purpose**

This software design document describes the system design of the Ballot Processor system, a system that determines the election results of Instant Runoff (IR) or Open Party List (OPL) elections. Developers working on this system, maintenance engineers, and testers are the intended audience of this document.

### **1.2 Scope**

This document covers the design of the full Ballot Processor system, from when an election official runs the program with an election data file, to the results determined and the audit file created by the program. An automated process to determine the results of an election is more efficient than determining the results by hand. An audit file produced by the system ensures our goal of full transparency and allows for the results to be validated by an election official.

### **1.3 Overview**

This document will describe the software system of our project and display the structure to complete the project. It will contain all requirements and the criteria our code will follow in the creation process.

### **1.4 Reference Material**

1. SRS Document for Group 5.
2. Use Case document for Group 5.
3. Project 1 Project Description.

### **1.5 Definitions and Acronyms**

IR: Instant Runoff

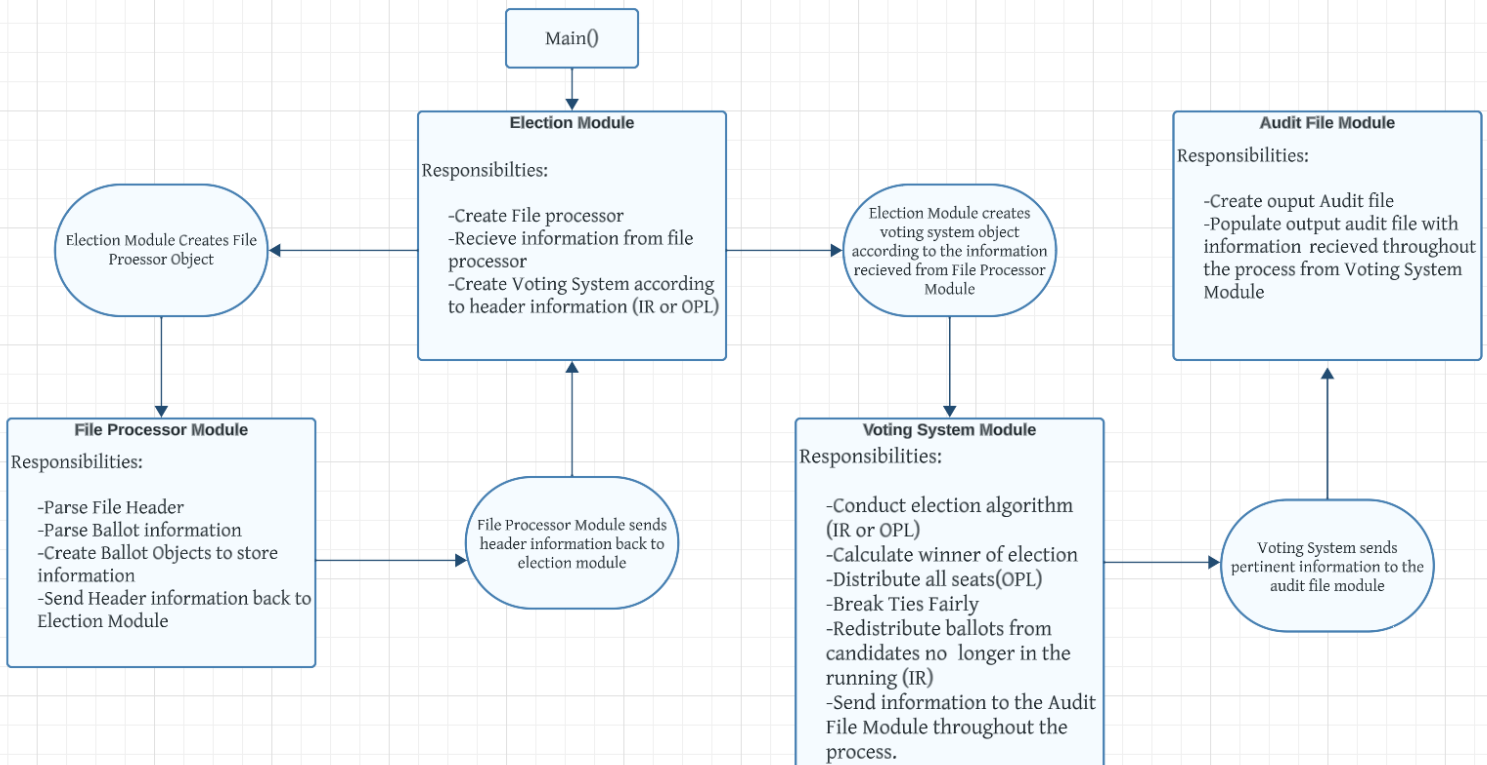
OPL: Open Party List

## **2. SYSTEM OVERVIEW**

This voting system functions as a processor for election data. It has the capability to determine the results of elections in the IR and OPL format. The system takes in a csv file containing election data, outputs an audit file containing the internal steps of the voting process and election results, and displays the final election results to the election official.

### 3. SYSTEM ARCHITECTURE

#### 3.1 Architectural Design



#### 3.2 Decomposition Description

The Election module will be where we create the file processor. This information in this is given from the file processor and can create an IR/OPL voting system from the information. The file processor module will take in the file header and parse through it to get the information about candidates, seats, parties, etc. This will also parse through the ballot information and create some objects for the ballot to store information. Once everything is parsed through, the header information will be sent to the election module. The voting system module will be the process that actually goes through the IR/OPL election process. This is where we find the winner and break ties. For OPL only, we will distribute the seats and for IR we redistribute ballots from removed candidates. Finally this module will send the calculated information to the file module. The audit file module has an end goal of creating an audit file which entails the winner. The information in the audit file is given from the previous voting system module.

### 3.3 Design Rationale

The architecture in 3.1 was selected due to the nature of the data available to the system. As ballots will be cast in separate locations and machines, the system needs to be able to input this data in the simplest form that can be transmitted from all ballot casters to an election official, and then read by the system: a CSV file. With this, the system creates an object that reads the data from the file.

Additionally, the audit of the election results must be able to be sent to multiple individuals and shared between different devices. With this, the program must interact with an output file, allowing a transferable audit file.

Finally, the election must be able to be counted in two different ways: OPL or IR. In order to accommodate this, the election component must be able to communicate with a separate voting system in order to specify which algorithm to use. The decomposition of these two components allows for this to happen.

## 4. DATA DESIGN

### 4.1 Data Description

Every ballot listed in the input file is represented with a Ballot object. Each candidate in the election is represented by a Candidate object, which stores the Ballots assigned to that Candidate in a List.

### 4.2 Data Dictionary

#### **AuditFile:**

- attributes:
  - outputFile: FileWriter
- methods:
  - + Audit()
  - + log(info:String):void

#### **Election:**

- attributes:
  - - fileName:string
  - - headerProcessor: HeaderProcessor
  - - votingSystem: VotingSystem
- methods:
  - + Election(fileName:string)

#### **File Processor:**

- attributes:
  - None
- method:
  - + processBallots(fileName:string)

#### **Header Processor:**

- attributes: None
- methods:
  - + HeaderProcessor():void
  - + parseHeader():VotingSystem

#### **IR Ballot:**

- attributes:
  - - votes:List<int>
  - - rank:int = 0
- methods:
  - + IRBallot(votes:List<int>):void
  - + getNextCandidate():int

#### **IR Candidate:**

- attributes:
  - - inRunning:bool = true
- methods:
  - + removeCandidate():void
  - + giveBallot(ballot: Ballot): bool

#### **IR Voting System:**

- attributes:none
- methods:
  - +runElection():void
  - -checkMajority(): bool

#### **OPL Ballot Party:**

- attributes:
  - - assignedto:int
- methods:
  - + OPLBallot(int):void

- + getAssignedTo():int

#### **OPL Candidate:**

- attributes:
  - -seated:bool=false
- methods:
  - +giveBallot(ballot: Ballot):bool
  - +seat():void

#### **OPL Voting System:**

- attributes:
  - - numberOfParties:int
  - - parties:List<Party>
- methods:
  - +assignSeats():void
  - +runElection(list<Party>): void

#### **Party**

- attributes:
  - -nameofParty: string
  - -candidates: List<OPLCandidate>
  - -totalVotes: int
  - -seatsAssigned: int
- methods:
  - +Party(name:string, candidates:List<OPLCandidate>): void
  - +rankCandidates(): void
  - +retrieveElected(): string
  - +giveCandidate(OPLCandidate): void

## **5. COMPONENT DESIGN**

#### **AuditFile:**

- methods:
  - + Audit()
    - Create audit object
    - Create file called Audit.txt or Audit.out in the same directory as the input file
    - Return pointer to the file and assign to the outputFile attribute
  - + log(info:String):void
    - Print 'info' string to the file pointed to by outputFile



**Election:**

- methods:
  - + Election(fileName:string)
    - Create election object
    - assign fileName passed in as the fileName attribute of this object
    - open file

**File Processor:**

- method:
  - + processBallots(fileName:string)
    - Iterate through the file line by line
    - create ballot objects according to the specification on each line in the file
    - Return when EOF is reached

**Header Processor:**

- methods:
  - + HeaderProcessor():void
    - Create Header Processor object
  - + parseHeader():VotingSystem
    - Iterate through file header line by line
    - Get election type on line one
    - Get number of candidates on line two
    - Get candidate names and parties on line three.
    - Create candidate objects for each candidate, filling in name and party and noting order.
    - Get number of ballots on line four.
    - Create voting system object of correct type and pass in number of candidates, list of candidate objects, and number of ballots

**IR Ballot:**

- methods:
  - + IRBallot(votes:List<int>):void
    - Create IRBallot object.
    - Initialize votes attribute to the list of ints passed in.
  - + getNextCandidate():int
    - Increase rank attribute by one
    - Return the number in the votes list at the index matching the rank attribute.

### **IR Candidate:**

- methods:
  - + removeCandidate():void
    - Iterate through the ballots list and call getNextCandidate() method on each.
    - Call giveBallot() method on candidate object corresponding to the int returned by getNextCandidate().
    - Change inRunning attribute to equal false.
  - + giveBallot(ballot: Ballot): bool
    - Add Ballot object passed in to the ballots list attribute.
    - Return true if success, return false if any failure occurs.

### **IR Voting System:**

- methods:
  - +runElection():void
    - Call checkMajority().
    - while(!checkMajority())
      - Call removeCandidate on the candidate object with the fewest ballots.
      - If two candidates are tied for last place, break the tie by randomly picking a winner.
      - Send information to audit object.
    - Print election results to terminal.
  - -checkMajority(): bool
    - Iterate through the list of candidates, checking size of the ballot list attribute for each candidate.
    - if (size of ballot list > 50% of total ballots)
      - return true
    - else
      - return false

### **OPL Ballot:**

- methods:
  - + OPLBallot(int): void
    - Creates an OPLBallot object.
    - Instantiates the assignedto attribute to be equal to the int passed in.
  - + getAssignedTo(): int
    - Returns the integer stored in the assignedto attribute.

**OPL Candidate:**

- methods:
  - +giveBallot(ballot: Ballot):bool
    - Adds ballot object passed in to the ballots list attribute.
    - Return true if success.
    - Return false if failure.
  - +seat():void
    - Changes the seated attribute to true.

**OPL Voting System:**

- methods:
  - +assignSeats():void
    - for( int i =0; i < numSeats; i++)
      - call retrieveElected() on the party
      - the return string from retrieveElected is the name of the candidate that is seated
  - +runElection(list<Party>): void
    - Divides the number of ballots by the number of seats and sets this as the quota.
    - Iterates through each party.
      - Iterates through each candidate.
        - Adds number of ballots in ballot list attribute to total number of ballots for this party.
    - Ballot total / quota = seats for that party. Ballot total % quota = remainder.
    - if no seats are left:
      - call assignSeats() on each party
      - Election is over.
      - Print results to the screen.
    - else
      - Parties with the largest remainders are given the remaining seats.
      - call assignSeats() on each party
      - Election is over.
      - Print results to the screen.

**Party**

- methods:
  - +Party(name: string, candidates: List<OPLCandidate>): void
    - Create a Party object.

- Instantiate nameofParty attribute to be equal to the string passed in.
- Instantiate candidates list to be equal to the OPLCandidate list passed in.
- +rankCandidates(): void
  - Iterate through candidates list attribute:
    - For each candidate call getBallotCount().
    - Order candidates list according to ballot count, highest to lowest.
- +retrieveElected(): string
  - Call seat() on first candidate in candidate list.
  - Return the first candidate in the candidate list
- +giveCandidate(OPLCandidate): void
  - Add OPLCandidate object to candidates list attribute.

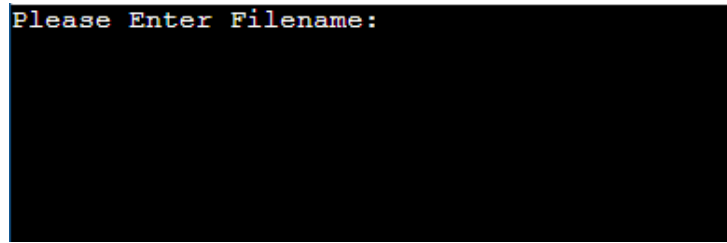
## 6. HUMAN INTERFACE DESIGN

### 6.1 Overview of User Interface

The user will interact with the system using a command line interface and will receive an audit file in the same directory as the program when the election results are completed. The audit file will describe to the user how the election results were generated.

In the beginning of the program, the system will prompt the user to enter the file name to the command line interface. Should the program not be able to open the file, then the system will notify the user that they need to either move the file noted into the same directory or enter a different file name.

### 6.2 Screen Images



### 6.3 Screen Objects and Actions

The program will have the terminal as the screen object and display the above image. This terminal will provide the users with inputting a filename as the action.

## 7. REQUIREMENTS MATRIX

Component / Data Structure	Requirement
Election Class	-UC_010: Program must have the functionality to switch off ballot shuffling, to

	ensure proper testing.
Header Processor Class	-UC_002: Program must be able to process the header and receive information such as election type, candidates, and number of ballots.
File Processor Class	-UC_001: Program must be able to open the valid .csv file that is passed by the user. -UC_011: Program must be able to process the ballot information held in the .csv input file, information regarding the votes cast, including ranking for IR, must be parsed and stored in the program.
Voting System Class	-UC_008: Program must be able to run an Instant Runoff election. -UC_009: Program must be able to run an Open Party List election. -UC_007: Program must display the number of seats in an OPL election. -UC_005: Program must be able to handle a tie in the election at anypoint in a fair way. -UC_006: Program must be able to handle a situation where there is no clear majority in IR. -UC_004: Program must print the election results directly to the screen for the user to get quick information regarding the results.
Audit File Class	-UC_003: Program must generate an audit file that presents detailed information about each step of the election process.

## 8. APPENDICES

Not applicable.