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[1] for the full IEEE Recommended Practice for Software Design Descriptions.

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

(Team 7)

(Project Aegis)

Software Design Document

Name (s): Michael Ly,
Donald Huynh,
Lucky Vang, Grant
Golemo

Lab Section: Workstation:

Date: 02/26/20201

TABLE OF CONTENTS

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

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 Aegis voting system project.

The intended audience is election officials, developers, and testers interested in the design and use of a voting system capable of performing Open Party Listing and Instant Runoff voting.

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 Aegis.

The basic architecture is a terminal window. The program will be in C++.

1.3 Overview

<<Provide an overview of this document and its organization. >>

The remaining chapters and their contents are listed below.

(Insert this later)

1.4 Reference Material

This section is optional.

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

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.

OPL	Open Party Listing
IR	Instant Runoff
C++	C plus plus language
Terminal Window	A program native to all computers that is used to run programs.

2. SYSTEM OVERVIEW

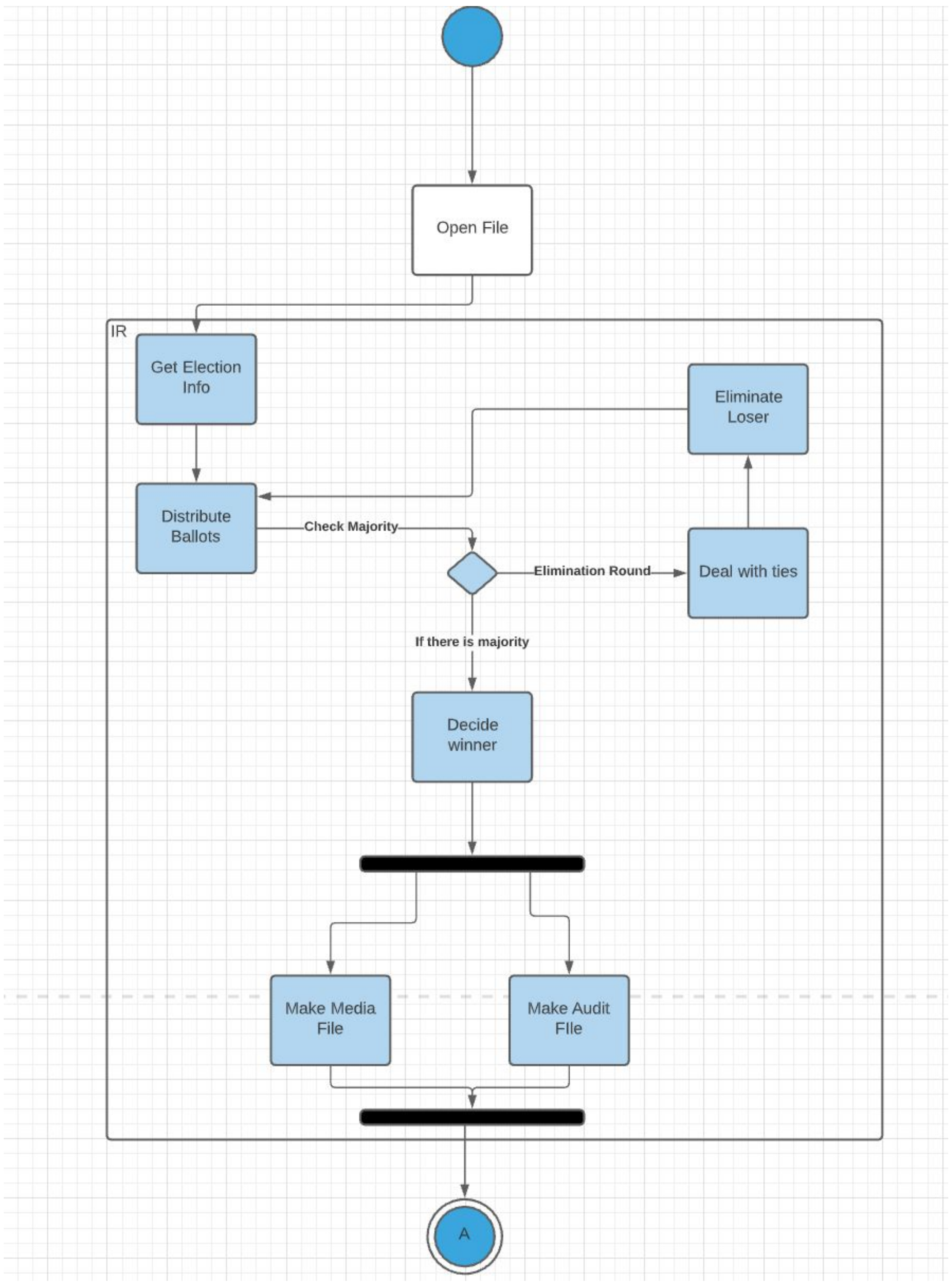
<< Give a general description of the functionality, context and design of your project. Provide any background information if necessary. >>

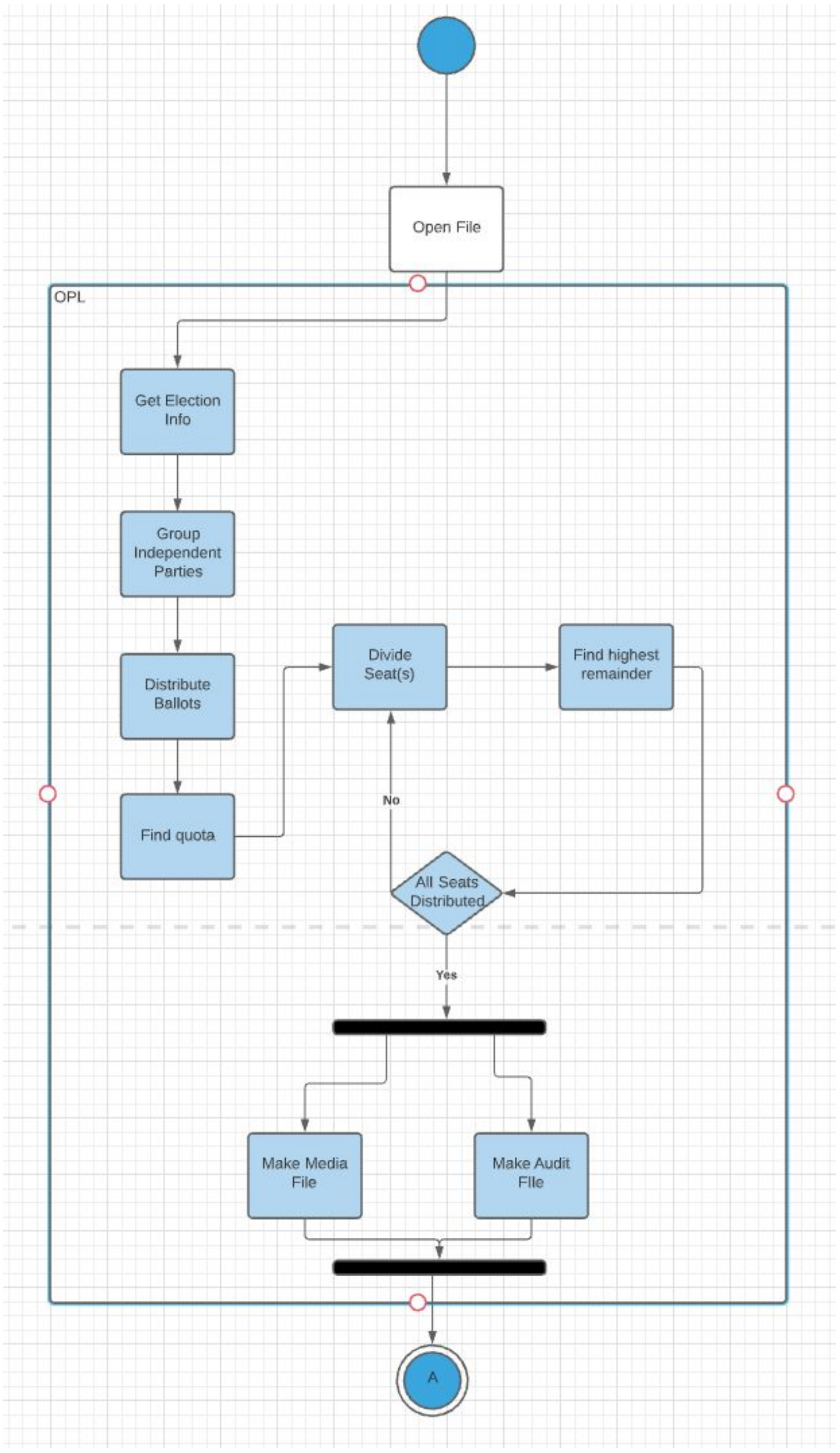
Project Aegis is intended to be a program, allowing for the process of elections run on the OPL and IR voting systems. The context of this project is to allow developers and students to explore software engineering and design of a project as well as provide a system to run elections.

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.




Open File
OPL
Get Election Info
Group Independent Parties
Distribute Ballots
Find quota
Divide Seat(s)
Find highest remainder
No
All Seats Distributed
Yes
Make Media File
Make Audit FIle
A

Voting System
Name: Voting System
Type: Class
Description: The voting system will be the highest super class and an interface. It will provide the methods and attributes for the entire system.
Attributes:

fileName: string
numBallots: int
timeTaken: double
candidates: vector<Candidate>

Operations:

Name: getNumBallots()
Arguments: None
Returns: The number of ballots
Pre-condition: None
Post-condition: Number of ballots are returned to the method that called it.
Exceptions: None.
Flow of Events:

1. The system needs a number of ballots
2. The system calls the method.
3. The system receives the number of ballots.

Name: getTimeTaken()
Arguments: None
Returns: The time taken to run the program start to finish
Pre-condition: The main functions of the system are run.
Post-condition: The time taken to run the program is returned
Exceptions: None.
Flow of Events:

1. The program has decided the election results
2. The program calls for the time taken.
3. The system returns the time taken to run the program.

Name: BreakTie()
Arguments: Candidate1, Candidate2, …. , CandidateN
Returns: One of the candidates.
Pre-condition: There is a tie in the number of votes between 2 or more candidates.
Post-conditions: One of the candidates will be determined the winner of the tie.
Exceptions: None
Flow of Events:

1. The system sees that there's a tie.
2. The system calls BreakTie in order to break the tie.

3. The system then receives a candidate from breakTie.

OPL
Name: OPL
Type: Class, inherits Voting System
Description: After reading and processing the inputted file, determine if OPL should be run and continue the program accordingly.
Attributes: n/a
Operations:

Name: runOPL()

Arguments:

file: FILE

Returns: int

Pre-condition: The file has been processed and OPL has been determined as the voting system type

Post-condition: The results of the election are displayed. -1 is returned if an error has occurred, otherwise 0 is returned.

Exceptions:

Flow of Events:

1. The system takes in a csv file
2. The system will process the file and determine which voting system to run.
3. The system now runs the calculations using the OPL algorithm.
4. The system will display the results of the election.
5. The system will produce an audit file and a media file in the same directory as the program.

IR
Name: IRL
Type: Class, inherits Voting System
Description: After reading and processing the inputted file, determine if IR should be run and continue the program accordingly.
Attributes: n/a
Operations:

Name: runIR()

Arguments:

file: FILE

Returns: int

Pre-condition: The file has been processed and IR has been determined as the voting system type
Post-condition: The results of the election are displayed. A -1 is returned if an error has occurred, otherwise a 0 is returned.
Exceptions:
Flow of Events:

1. The system takes in a csv file
2. The system will process the file and determine which voting system to run.
3. The system now runs the calculations using the IR algorithm.
4. The system will display the results of the election.
5. The system will produce an audit file and a media file in the same directory as the program.

Name: elimination()
Arguments:

Candidate
Name: Candidate
Type: Class
Description: An object designed to store all necessary information about a candidate.
Attributes:
votes: vector<Ballots>
party: char
numVotes: int
name: string
Operations:
Name: getParty()
Arguments: none
Returns: char
Pre-condition: a candidate object has been instantiated
Post-condition: the party type is returned
Exception:
Flow of Events:

1. There has been a call to getParty()
2. The party of the calling candidate is returned

Name: getNumVotes()
Arguments: none
Returns: int
Pre-condition: a candidate object has been instantiated
Post-condition: the number of votes is returned
Exception:
Flow of Events:

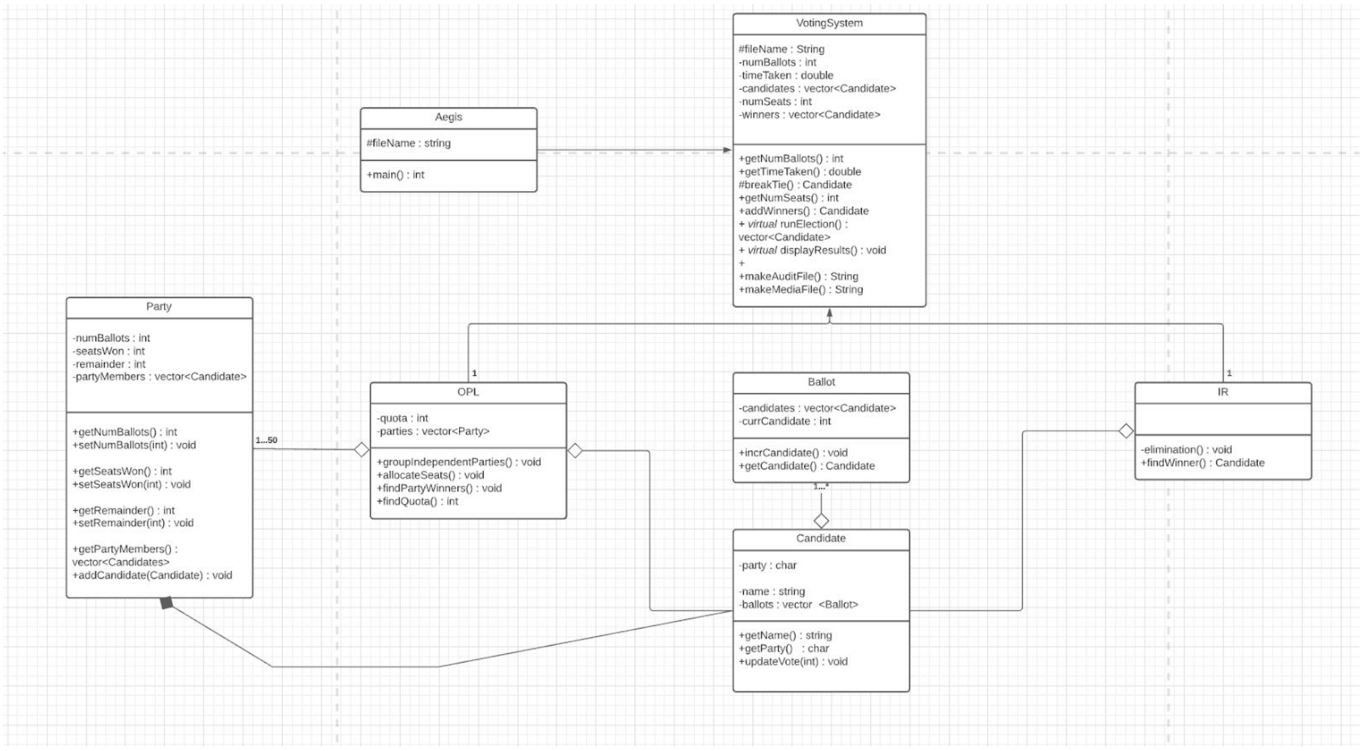
1. There has been a call to getNumVotes()
2. The number of votes of the calling candidate is returned

Name: updateVotes()
Arguments:
newVote: int
Returns: void
Pre-condition: a candidate object has been instantiated
Post-condition: the number of votes has been changed
Exception:
Flow of Events:
1. There has been a call to updateVotes(int newVote)
2. The number of votes has been set to newVote

Ballot
Name: Ballot
Type: Class
Description:
Attributes:
votes: vector<Candidate>
Operations:
Name:getCandidates()
Arguments: none
Returns: vector<Candidate>
Pre-condition: a ballot object has been instantiated
Post-condition: the candidates is returned
Exception:
Flow of Events:
1. There has been a call to getCandidates()
2. The candidates is returned

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 a 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.



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.

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.

All ballots are taken in through a CSV file. Each line past the first 3 represents one ballot. Each of these lines is fed into a constructor that will turn the information into a ballot object, which contains the candidates that were voted for and in which order. These ballots are all stored in the vector to which it's top vote belongs to.

Each candidate in an election has its own candidate object, along with a vector full of the votes that currently belong to them.

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.

Voting System
fileName: string
numBallots: int
timeTaken: double
candidates: vector<Candidate>

Candidate
party: char
numVotes: int
name: string

Ballot
votes: vector<Candidate>

Attribute Name	Attribute Type	Attribute Size
candidates-	vector<Candidate>	261+vector
fileName#	String	256
name-	String	256
numBallots-	Int	4
numVotes-	Int	4

party-	Char	1
timeTaken-	Double	8
votes	vector<Candidate>	

Symbol	**Meaning**
+	public
-	private
#	protected
~	package

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.

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.

Election Official:

In order to use the Aegis program, the user will have to download the program. Next, open the terminal and type the following command without the quotes: "./Aegis file.csv". This will signal the program to execute. No other input from the user is required for the program to function properly. The results of the election will be displayed on the screen and both an audit and media file will be produced and stored inside the same directory as Aegis.

Developer/Tester:

In order to use the Aegis program, the user will have to download the program. Next, open the terminal and type the following command without the quotes: "./Aegis file.csv" along with any flags that the user wants to trigger. This will signal the program to execute and run certain commands depending on the inputted flags. The results of the election will be displayed on the screen and both an audit and media file will be produced and stored inside the same directory as Aegis.

6.2 Screen Images

Display screenshots showing the interface from the user's perspective. These can be hand drawn or you can use an automated drawing tool. Just make them as accurate as possible. (Graph paper works well.)

6.3 Screen Objects and Actions

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

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.

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.