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 12

IR and OPL Voting System

Software Design Document

Names: Max Bryson, Ruichen He, Hannah Nelson, Christopher Miao

Lab Section:

Workstation:

Date: (10/10/2023)

TABLE OF CONTENTS

1. Introduction

1.1 Purpose

This software design document describes the architecture and system design of a voting system that can decide a winner through two different voting types, open party listing (OPL) and instant runoff (IR). This document is intended for developers and managers.

1.2 Scope

The software will be able to take in a csv formatted file that holds the votes for an election and return the winner. The users will have the option of using either OPL voting or IR voting. The software will be able to process 100,000 ballots in under 8 minutes which will aid the election officials in producing a winner in a timely manner for the public. The system will not tamper with the ballots which will ensure that the winner is declared fairly and accurately. Upon finding a winner, the system will produce an audit file where the process of counting the votes and declaring a winner will be written.

1.3 Overview

This document begins with an introduction for the readers to familiarize themselves with the subject and terminology used within the proposed system. This is followed by a general overview of the system. After which several designs for the system are depicted. The first design describes how the system is organized at the hardware and software level. The second design covers how data is kept and managed within the system. The third design details the finer organization of each component within the architecture. Finally, the user-interface design is detailed. After the diagrams, a table is created to map use-cases from the software requirements specification documents to items in the aforementioned diagrams.

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

OPL stands for open-party list. This is a type of election where the voter's vote counts first for the party and second for the candidate within the party. The winners are decided by how many votes their parties got, then by how many votes within their party they got.

IR stands for instant runoff. This is a type of election where the voter ranks each candidate based on their preferred winner. The winner is determined by majority vote. If, at the end of counting, there is no majority, the lowest voted candidate is dropped and their votes are redistributed to the next ranked candidate.

2. System Overview

This project is meant to determine the winner of an election and to output an audit file showing how the election progressed based on the ballot that was generated from an election official. The program will be able to function for either an IR or OPL election type. The audit file will be slightly different for IR and OPL elections. In IR, it will show how the votes get redistributed after each cycle of the election. In OPL, it will show how many seats each party receives and which candidates will receive them. In the case of a tie, a fair coin toss will determine the winner. This program is expected to be run multiple times a year for normal and special elections. C++ is the language that was chosen for this project.

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.

The designed system adopts an Object-Oriented (OO) architecture, featuring a series of interrelated classes, each embodying distinct responsibilities and behaviors. Central to this system is the Election class, acting as the primary orchestrator of the election process. Initiating the sequence of events is the Main class, which sets the election in motion by interacting with the Election class. Within its purview, the Election class manages crucial details pertaining to the election options, the associated file data, and the overall tally of votes. This class collaboratively works with the DataStructure class to handle foundational attributes of the options, notably their names and the vote counts. To accommodate varied election methodologies, the architecture incorporates the IR and OPL classes. The IR class, symbolizing the Instant Runoff approach, incorporates the IRCandidate class to represent individual contestants. This encapsulation ensures the efficient tracking of votes and the real-time status of each candidate. In contrast, the OPL class, an embodiment of the Open Party List system, interfaces with the OPLParty class. This liaison effectively manages parties, their associated members, and the votes they garner.

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 objectoriented description. For a functional description, put toplevel 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 Election class serves as the primary interface for the two election systems: IR and OPL. This class holds data members such as options, filename, fp, and totalVotes. It also provides methods like Run() and CoinFlip() to execute election processes. The DataStructure class holds basic details for different election entities, primarily focusing on their names and votes. The IR class, dedicated to the Instant Runoff election system, contains the filename and offers methods like Run() and findMajority(). This class works closely with the IRCandidate class, which maintains vote patterns and determines the candidacy status. On the other hand, the OPL class focuses on the Open Party List system. It contains data attributes like numSeats and quota and offers functions such as Run() and findQuota(). The OPLParty class complements the OPL class, managing party details, member information, and vote counts.

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 architecture chosen for this system is rooted in an Object-Oriented (OO) design, influenced by the inherent hierarchical and interrelated nature of real-world electoral processes. By employing an OO approach, the system ensures modularity and scalability, with distinct classes like Election, IR, OPL, and their associated entities. This design choice promotes clarity by representing entities such as elections, candidates, and parties in an intuitive manner. Furthermore, the introduction of foundational classes, such as DataStructure, aids in reducing redundancy and streamlining operations, ensuring that the system remains adaptable and maintainable.

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.

Our system is designed to process election data, converting raw voting records into structured data that supports two types of election systems: IR and OPL. The transformation begins when raw data is ingested by the Election class, which serves as the primary interface. Through the various classes and their methods, the information domain is organized into well-defined data structures to efficiently run election processes.

The data is stored primarily in in-memory structures, represented by the various classes. There isn't a standalone database integrated, but the system does rely on file handling mechanisms, where data like voting records and results can be read from or written to files.

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.

1. DataStructure Class (Base Class)
    a. Attributes:
        i. Names: Name of election entities.
        ii. numVotes: total number of votes for candidate or party
    b. Methods: (None specified)
        i. GetNumVotes(): Calculate the vote count
        ii. GetName(): Return the name of the candidate or party
2. Election Class (Base Class)
    a. Attributes:
        i. Options: A list of DataStructure objects
        ii. filename: Name of the file containing voting records.
        iii. fp: The real file content
        iv. totalVotes: Total votes counted.
    b. Methods:
        i. Election(filename: String): Initialize the Election class
        ii. Run(): A virtual function
        iii. CoinFlip(): Determines the winner in case of a tie.
3. IR Class (Derived Class, from Election Class)
    a. Methods:
        i. IR(): Constructor functions for initiating the attribute values.
        ii. Run(): Executes the Instant Runoff election process.
        iii. findMajority(): Determines the majority candidate in the IR election.
4. IRCandidate Class
    a. Attributes:
        i. votes: Sequences of votes for the candidate.
        ii. inRace: True or false whether the candidate is in the current round or not.
    b. Methods:
        i. IRCandidate(name: String): Initialize a new IRCandidate class with the candidate name.
        ii. AddVote(vote: vector<int>): Add a vote that vote the candidate at the first place
        iii. Reallocate(): Reallocate the votes based on the following candidates on the vote

    iv. GetInRace():Getter function to return if the candidate is in the race or not.
5. OPL Class
    a. Attributes:
        i. numSeats: Number of seats available in the OPL election.
        ii. quota: Number of seats available in the OPL election.
    b. Methods:
        i. OPL(): Constructor functions for initiating the attribute values.
        ii. Run(): Executes the Open Party List election process.
        iii. findQuota(): Determines the quota for the OPL election.
6. OPLParty Class
    a. Attributes:
        i. memberList: Details of the party participating in the OPL election.
    b. Methods:
        i. OPLParty(name: String): Initialize a new OPLParty class with the party name.
        ii. AddMember(member: String): Add party member.
        iii. AddVote(): Add vote that belongs to the party.
        iv. getMemberList(): Get the list of the members belonging to the party.

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.

1. IR Class (Derived from Election Class)
    a. Run()

```
WHILE no candidate has majority AND candidates are left:

   CALL findMajority()

   IF no candidate has majority THEN
```

```
        ELIMINATE candidate with least votes
        FOR each candidate in IRCandidate:
            CALL candidate.Reallocate()
        END FOR
    END IF
END WHILE
PRINT winning candidate
```

b. findMajority()

```
FOR each candidate in IRCandidate:
    CALCULATE total votes
    IF votes > 50% of totalVotes THEN
        RETURN candidate
    END IF
END FOR
RETURN NULL
```

2. OPL Class
   a. Run()

```
OPEN filename
READ voting records for parties
CALCULATE total seats based on votes
FOR each party in OPLParty:
    ASSIGN seats based on findQuota()
END FOR
PRINT results
```

b. findQuota()

```
CALCULATE quota based on totalVotes and numSeats
RETURN quota
```

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 will enter the filename from a prompt in the terminal, the system will then reprompt if the file could not be found or process the file. The information that shows the step by step process for finding the winner will be displayed on the terminal as well as printed to an audit file.

6.2 Screen Images

A text mockup of the UI:

```
Enter Filename: USER INPUT

Election is IR.

Round 1

Candidates & Parties:       Votes:

Susan Rosen (Dem.)          43,000

Nina Kleinberg (Rep.)       42,000
```

Thomas Choi (Ind.)	8,000
Edward Royce (Libert)	7,000

Round 2 …

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.

UC_001	Main.
UC_002	Main
UC_003	Main
UC_004	AuditSystem()
UC_005	OPL/IR, run(), uses dataStructure
UC_006	CoinFlip()
UC_007	IR, run(), also uses IRCanidate
UC_008	IR, run(), findMajority() also uses IRCanidate
UC_009	IRCanidate, reallocate()
UC_010	OPL, FindQuota(), also uses OPLParty
UC_011	OPL, run(), also uses OPLParty