

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.ieee.org/standards/publications/10161998.html)¹ for the full IEEE Recommended Practice for Software Design Descriptions.

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

(Team Name)

(Project Title)

Software Design Document

Name (s):

Lab Section:

Workstation:

Date: (mm/dd/yyyy)

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 document describes the complete detailed structure of the CompuVote election system and its architecture as a part of the implementation of the requirements specified in the Software Requirements Specification (SRS) document. The document provides insights on the data structures and algorithms the CompuVote election system employs. It also assists in identifying the different systems involved in CompuVote and how they interact with each other. This Software Design Description is written for knowledgeable software professionals and architects. This includes developers, testers, and documentation writers.

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.

CompuVote is a program developed in Java that is responsible for counting votes to get election results for two voting systems: Instant Runoff Voting and Open Party List Voting. The purpose of the product is to provide a program that can tally and compute the winning candidates for the Instant Runoff and Open Party List voting systems and includes the infrastructure to easily support the addition of other voting systems. The system can be used for any situation that involves elections and requires tallying ballots compiled into the specified CSV format. Businesses or corporations can also use CompuVote for their own internal voting systems. CompuVote generates an audit file that explains the entire process used to compute the winners of an election in thorough detail and generates a report file that displays a summary of the election.

1.3 Overview

Provide an overview of this document and its organization.

- Section 2
- Section 3 provides detailed descriptions and visual diagrams of the architectural design of CompuVote.
 - 3.1: The main purpose is to allow the reader to gain a general understanding of the overall architecture of the system. It identifies each high level subsystem and the roles or responsibilities assigned to it. It also explains the relationships between the modules to achieve the complete functionality of the system.

- 3.2: This section provides a decomposition of the systems in the architectural design in CompuVote. Sequence and activity diagrams showing the flow of the program are given.
 - 3.3
- Section 4 describes the data design within CompuVote
 - 4.1 details how the data from the election files are extracted and how they are stored in the data structures.
 - 4.2 lists the system entities or major data along with their types and descriptions.
- Section 5 provides a summary of the algorithm for each function listed in 4.2 in pseudocode.

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.

2. SYSTEM OVERVIEW

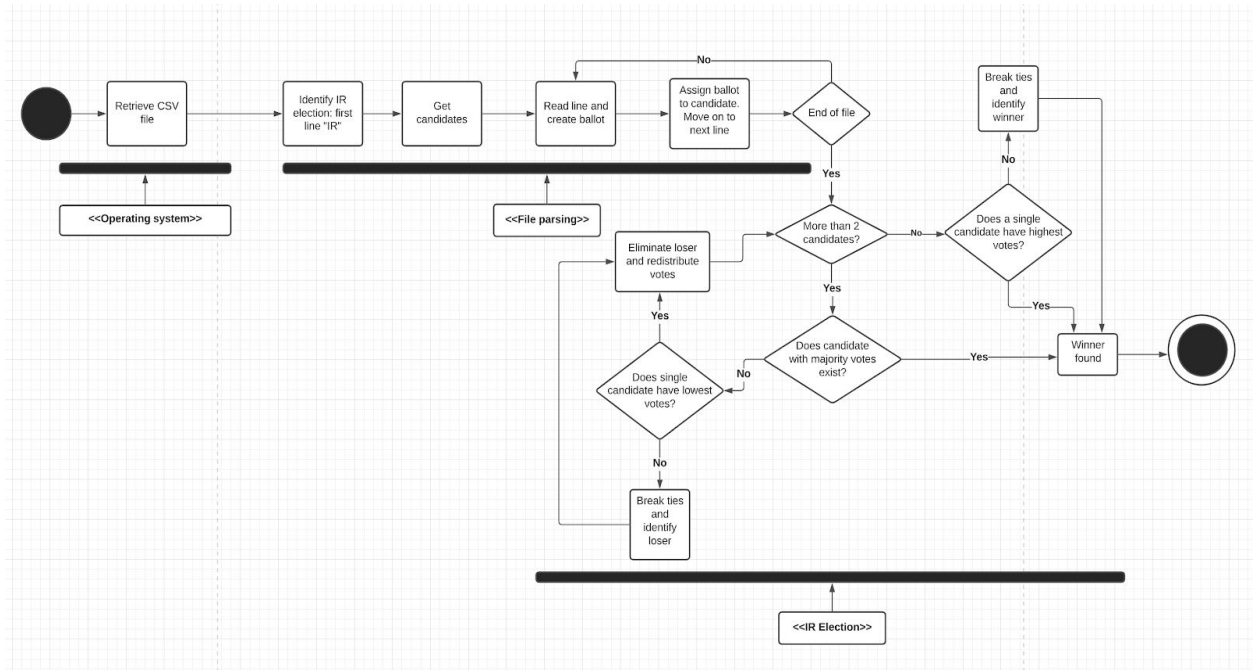
Give a general description of the functionality, context and design of your project. Provide any background information if necessary.

3. SYSTEM ARCHITECTURE

3.1 Architectural Design

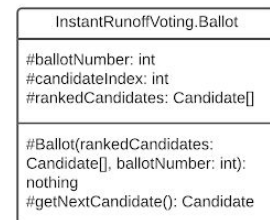
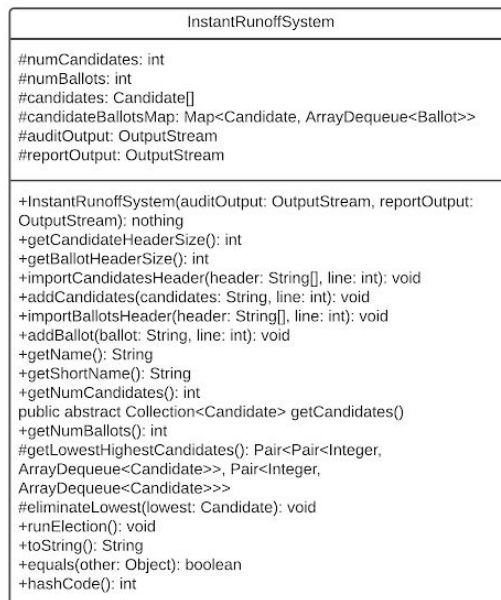
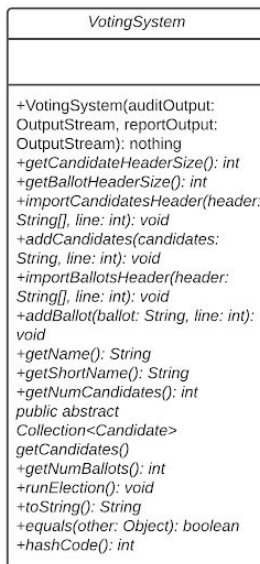
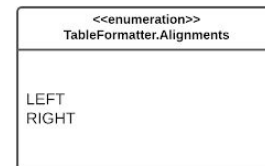
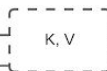
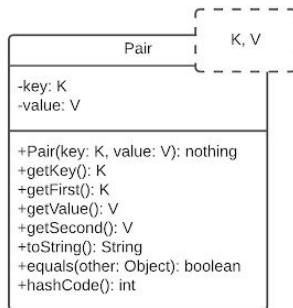
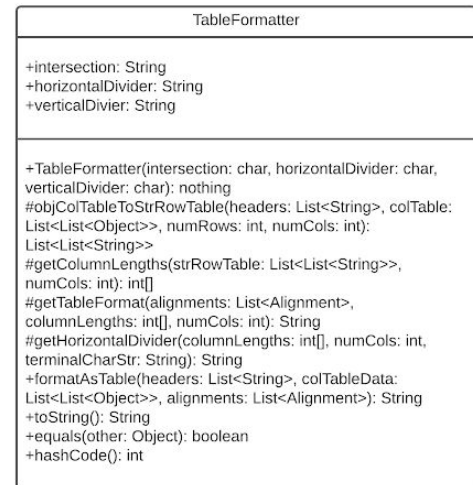
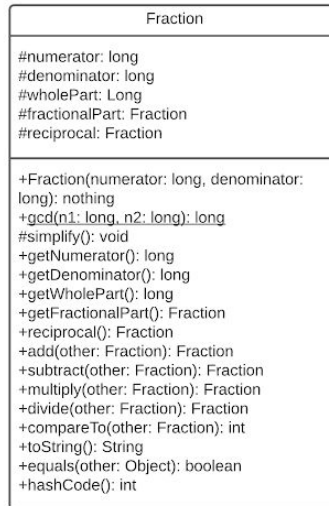
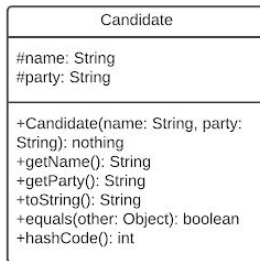
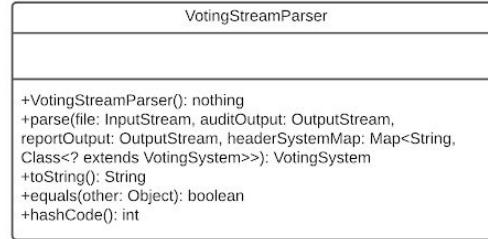
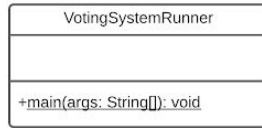
Develop a modular program structure and explain the relationships between the modules to achieve the complete functionality of the system. This is a high level overview of how

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.



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.

The input to the system is a Comma-Separated-Values File (CSV file). Once the file is read into the system, it must be parsed in order to extract the data. Depending on the system, there will be a slightly different parsing algorithm. For both systems, the second line is parsed as a nonnegative integer, the third line is parsed as a list of candidates and their corresponding parties, the fourth line is parsed as a nonnegative integer. There will also be an additional number of lines equivalent to the number of ballots (the number returned from the fourth line). These satisfy the requirements for an Instant Runoff Election. For Open Party Listings, the fifth line is parsed as a nonnegative integer. All of the values that were parsed are saved as variables and can be used within the program.

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.

System Entity	Type	Description
AllocateInitialSeats(int quota, int partyVotes)	int	Using the calculated quota, the initial number of seats is calculated for each party by dividing the number of party votes by the quota (truncating the number of seats).
AllocateRemainingSeats(Array remaining[])	int	Compare the remainder values from

		AllocateInitialSeats(). A second allocation of seats will occur, where the next subsequent party with the next most votes is granted the seat. Returns the total number of seats allocated for each party.
BreakTieIr(Array candidates[])	string	Randomly picks a loser when 2 or more candidates have the same number of votes. Returns the chosen candidate.
BreakTieOpl(Array candidates[])	string	Breaks a tie in the OPL voting system. Can be incited from AllocateRemainingSeats() or DistributePartySeats(). Returns the chosen party/candidate.
CalculateQuota(int numVotes, int unfilledSeats)	int	The quota is calculated (and returned upon completion) which is the total number of votes cast divided by the number of seats to be filled.
CountCurrentRank(Array candidates[])	void	Identifies that candidate with majority votes and candidate(s) with the lowest ballot count if no candidate has over 50% of the total votes.
DeclareSeats()	void	The seats allocated to parties and candidates will be declared.
DeclareWinner(Array candidates[])	string	Declares a final winner of the IR election.
DeterminePopularityWinner (Array candidates[])	string	In the event that all eliminations have taken place (in IR) and no candidate has a majority, the candidate with the most votes wins.

DistributePartySeats(Array partiesAndCandidates[[]], Array filledSeats[])	void	Distributes the seats for each party to the candidates with the highest votes in order.
EliminateLowest(Array candidates[])	void	Eliminates the lowest ranked candidate.
GroupIndependents(Array candidates[], Array partiesAndCandidates[[]])	void	Groups all independent candidates into one party.
GroupParties(Array candidates[], Array partiesAndCandidates[[]])	void	Groups all candidates into their respective party.
IdentifyVotingSystem(FILE *electionFile)	string	The system must parse the file and determine the election type. Will return “IR” or “OPL”.
ParseFile(FILE *electionFile)	void	The system must parse the file for the information required to run the election in an IR election.
ParseIrBallots(FILE *electionFile)	VotingSystem	The system must parse the file for the ballots in an IR election. The election system will be returned.
ParseIrCandidates(FILE *electionFile)	string[]	The system must parse the file for the candidates in an IR election. An array of candidates is returned.
ParseNumberOfSeats(FILE *electionFile)	void	The system must parse the number of seats for an IR election.
ParseOplBallots(FILE *electionFile)	VotingSystem	The system must parse the file for the ballots for an OPL. The election system will be returned.
ParseOplCandidates(FILE *electionFile)	string[]	The system must parse the file for the candidates for an OPL. An array of candidates is returned.

PresentSummaryToScreen()	void	<p>A summary of the election will be printed to the screen upon completion of the entire program. This will include the following:</p> <ol style="list-style-type: none"> 1. Type of election 2. Number of candidates 3. The candidates and their parties 4. Number of ballots 5. Number of seats (only for OPL) 6. The quota (only for OPL) 7. How many votes each candidate had 8. The percentage of votes each candidate had
RetrieveFile(FILE *electionFile)	InputStream()	The system must be able to retrieve and read the file provided.
RunIrElection(Array ballots[], Array candidates[])	void	Runs the algorithm in an instant runoff voting format.
RunOpIElection(Array ballots[], Array candidates[], Array partiesAndCandidates[][])	void	Runs the algorithm in an open party listing format.
TransferBallots(Array ballots[], Array candidates[])	void	Transfer the eliminated candidate's ballots to the next ranked candidate.
WriteAuditToFile()	void	<p>An audit file will be produced upon completion of the program. This will include the following:</p> <ol style="list-style-type: none"> 1. Gives in-detail explanation of all of the steps that occurred in the election. 2. Tallying of each individual ballot 3. Type of election 4. Number of candidates

		<ol style="list-style-type: none"> 5. The candidates and their parties 6. Number of ballots 7. Number of seats (where applicable) 8. The quota (where applicable) 9. Every calculation that occurs during the election 10. How many votes each candidate had 11. The percentage of votes each candidate had
WriteReportToFile()	void	<p>A formatted report will be produced upon completion of the program. This will include the following:</p> <ol style="list-style-type: none"> 1. Type of election 2. Number of candidates 3. The candidates and their parties 4. Number of ballots 5. Number of seats (only for OPL) 6. The quota (only for OPL) 7. How many votes each candidate had 8. The percentage of votes each candidate had

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

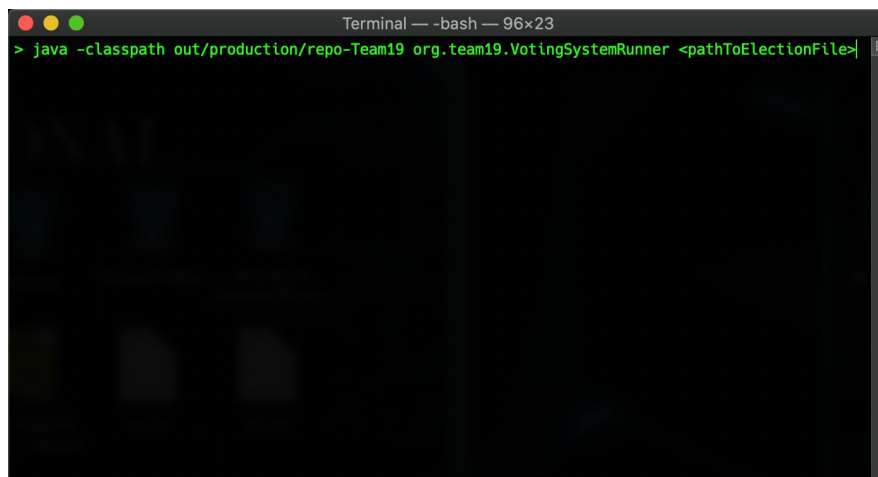
Describe the functionality of the system from the user's perspective. Explain how the user will be able to use your system to complete all the expected features and the feedback information that will be displayed for the user.

The user, who is person compiling and inputting the CSV file, will be doing so through CLI (command line interface) through the appropriate operating system (explanation for usage per OS in section 3.3 of the Software Requirements Specification Document). The file path to the election file is the only command-line argument required for the program to run. In the repo-Team19 directory, one runs the command `java -classpath out/production/repo-Team19 org.team19.VotingSystemRunner <pathToElectionFile>` to run the program where `<pathToElectionFile>` is replaced with the file path to the election file.

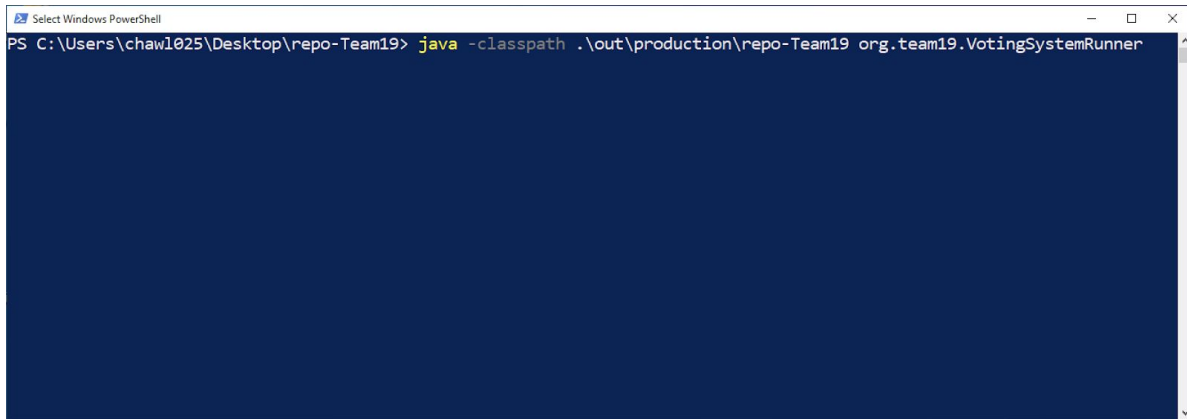
6.2 Screen Images

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.)

macOS Terminal:



Windows PowerShell:



```
PS C:\Users\chaw1025\Desktop\repo-Team19> java -classpath .\out\production\repo-Team19 org.team19.VotingSystemRunner
```

Ubuntu gnome-terminal:



```
> java -classpath out/production/repo-Team19 org.team19.VotingSystemRunner <pathToElectionFile>
```

6.3 Screen Objects and Actions

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

CLI:

- *Exit*: Closes Terminal.
- *Maximize/Store Down*: Maximize the size of the window/restore to original size.
- *Minimize*: Temporarily closes the terminal window, but can be reopened in the same state.
- *Scroll Bar*: A widget that allows one to scroll in a predetermined direction.

Ubuntu CLI:

- File
 - *New Tab*: Opens a new tab within the current terminal window.
 - *New Window*: Opens a new terminal window.
 - *Close Tab*: Closes the current tab.
 - *Close Window*: Closes the current window.
- Edit

- *Copy*: Makes a copy of the current highlighted message.
- *Copy as HTML*: Makes a copy of the current highlighted message in HTML format.
- *Paste*: Paste the current copied contents.
- *Select All*: Highlights all contents in the terminal.
- *Preferences*: Changes the terminal preferences for a given profile.
- View
 - *Show Menubar*: Checkbox that will determine if the menubar (bar with File, Edit, View, Search, Terminal, and Help) is displayed.
 - *Full Screen*: Maximizes the window to cover the entire screen.
 - *Zoom In*: Increases text size.
 - *Normal Size*: Returns to 100% zoom.
 - *Zoom Out*: Decreases text size.
- Search
 - *Find...*: Opens a search box to find a given word/phrase.
 - *Find Next*: Finds the next instance of a given word/phrase.
 - *Find Previous*: Finds the previous instance of a given word/phrase.
 - *Clear Highlight*: Clears currently highlighted text
- Terminal
 - *Read-Only*: Sets the mode to Read-Only. The user cannot input anything directly into the command line.
 - *Reset*: Fixes previous errors after escape sequences.
 - *Reset and Clear*: Fixes previous errors after escape sequences and clears the entire terminal of text.
- Help
 - *Contents*: Opens a window about how to work with Terminal. Gives a general introduction, as well as hyperlinks to other helpful features.
 - *About*: Opens a window that gives the information about GNOME Terminal, such as the current version, copyright, and a link to the website.

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.

Function ID	Existence in Pseudocode
UC_001	if len(args) == 0: inputStream = STDIN

	<pre> else if len(args) == 1 try: inputStream = retrieveInputStream(args[0]) except: presentError() else: presentError() </pre>
UC_002	<pre> if len(args) == 0: inputStream = STDIN else if len(args) == 1 try: inputStream = retrieveInputStream(args[0]) except: presentError() else: presentError() </pre>
UC_003	<pre> public class VotingStreamParser: public VotingStreamParser(): public VotingSystem parse(InputStream file, OutputStream auditFile, OutputStream reportFile, </pre>
UC_004	<pre> lineNumber = 1 try: firstLine = file.getLine() except: presentError(lineNumber) votingSystemClass = headerSystemMap.get(firstLine) </pre>
UC_005	<pre> candidateHeaderSize = votingSystem.getCandidateHeaderSize() try: candidatesHeader = file.getLines(candidateHeaderSize) except: presentError(lineNumber) try: votingSystem.importCandidatesHeader(candi datesHeader, lineNumber) </pre>

	<pre> except: presentError(lineNumber) lineNumber += candidateHeaderSize numCandidates = votingSystem.getNumCandidates() try: candidatesLine = file.getLine() except: presentError(lineNumber) try: votingSystem.addCandidates(candidatesLine) except: presentError(lineNumber) if len(votingSystem.getCandidates()) != numCandidates: presentError(lineNumber) </pre>
UC_006	<pre> ballotHeaderSize = votingSystem.getBallotHeaderSize() try: ballotHeader = file.getLines(ballotHeaderSize) except presentError(lineNumber) try: votingSystem.importBallotHeader(ballotHeader, lineNumber) except: presentError(lineNumber) lineNumber += ballotHeaderSize numBallots = votingSystem.getNumBallots() ballotNumber = 1 foreach remaining line: try: votingSystem.addBallot(line, lineNumber, ballotNumber) except: presentError(lineNumber) lineNumber++ ballotNumber++ </pre>

	<pre> if numBallots != ballotNumber - 1: presentError(lineNumber - 1) return votingSystem </pre>
UC_007	<pre> candidateHeaderSize = votingSystem.getCandidateHeaderSize() try: candidatesHeader = file.getLines(candidateHeaderSize) except: presentError(lineNumber) try: votingSystem.importCandidatesHeader(candi datesHeader, lineNumber) except: presentError(lineNumber) lineNumber += candidateHeaderSize numCandidates = votingSystem.getNumCandidates() try: candidatesLine = file.getLine() except: presentError(lineNumber) try: votingSystem.addCandidates(candidatesLine) except: presentError(lineNumber) if len(votingSystem.getCandidates()) != numCandidates: presentError(lineNumber) </pre>
UC_008	
UC_009	<pre> ballotHeaderSize = votingSystem.getBallotHeaderSize() try: ballotHeader = file.getLines(ballotHeaderSize) except presentError(lineNumber) try: votingSystem.importBallotHeader(ballotHead er, lineNumber) except: presentError(lineNumber) </pre>

	<pre> lineNumber += ballotHeaderSize numBallots = votingSystem.getNumBallots() ballotNumber = 1 foreach remaining line: try: votingSystem.addBallot(line, lineNumber, ballotNumber) except: presentError(lineNumber) lineNumber++ ballotNumber++ if numBallots != ballotNumber - 1: presentError(lineNumber - 1) return votingSystem </pre>
UC_010	<pre> public class InstantRunoffSystem extends VotingSystem: public void runElection(): </pre>
UC_011	<pre> if candidateBallotsMap.length() <= 2: ... else ... </pre>
UC_012	<pre> protected void eliminateLowest(Candidate lowest): </pre>
UC_013	<pre> if candidateBallotsMap.length() <= 2: ... else print randomlySelect(firstCandidate, secondCandidate) </pre>
UC_014	<pre> for ballot in ballotsToRedistribute: Candidate nextCandidate = ballot.getNextCandidate() if nextCandidate != UNDEFINED: candidateBallotsMap.get(nextCandidate).add(ballot) </pre>

UC_015	
UC_016	Candidate winner int firstSecondCandidateComparison = candidateBallotsMap.getFirst().compare(candidateBallotsMap.getSecond()), if firstSecondCandidateComparison > 0: print firstCandidate else if firstSecondCandidateComparison < 0: print secondCandidate else: print randomlySelect(firstCandidate, secondCandidate)
UC_017	public class OpenPartyListingSystem extends VotingSystem: void runElection():
UC_018	public void addCandidates(String candidates, int line):
UC_019	public void addCandidates(String candidates, int line):
UC_020	Fraction quota = Fraction(numBallots, numSeats)
UC_021	int allocateInitialSeats(Fraction quota):
UC_022	void allocateRemainingSeats():
UC_023	
UC_024	ArrayDeque<Candidate> distributeSeatsToCandidates():
UC_025	ArrayDeque<Candidate> distributeSeatsToCandidates():
UC_026	public String formatAsTable(List<String> headers, List<List<Object>> colTableData, List<Alignments> alignments):
UC_027	this.auditOutput = auditOutput
UC_028	this.reportOutput = reportOutput

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.