Sihang Li

**Business Requirements Specification**

**Introduction**

The software is a stand-along text file editor. Its main functionality is to provide a graphical user interface window (GUI) that allows the user to open up and edit a text file with extension .txt or .rtf. (“text file” hereafter only referrers to .txt and .rtf files) This software is intended to serve customers who need to find the frequency of a word in a text file for statistics, as well as the customers who need to simultaneously replace or delete all occurrence of a word in a text file.

**User Requirements Specification**

1. The system shall have some kind of user interface.
2. The system shall allow the system user to open up a .txt or .rtf file and the content of the file shall be displayed in the GUI window.
3. The system shall allow the user to directly modify the text in the display window and save changes directly to the original file.
4. The system shall allow the system user to find the frequency of a word in the opened file and it shall check if the word is an English word.
5. The system shall allow the system user to replace all the occurrence of a word in the opened file. “and it shall only replace those of the same case. In addition, the system it shall check if the word to be replaced and the word to replace are English words.
6. The system shall allow the system user to delete all the occurrence of a word in the opened file. In addition, the system shall check if the word is an English word.
7. The system shall have a help document that explains in detail how the system works.
8. The system shall terminate correctly by means of clicking on the “Exit” sub-tab or closing the GUI.
9. The system shall handle all exceptions.

**Structured Requirements table (table 1):**

|  |  |
| --- | --- |
| Text File Editor | |
| Function | Edit text file. |
| Description | Utilizes a graphical user interface which allows the user to open a .txt or .rtf file, find the number of a word as well as replace or delete a word in that file. Exits the program if the user clicks on the “Exit” button or closes the window. |
| Inputs | .txt or .rtf file (file), changes made to the file (c0), local English dictionary(e0), word to find (f0), word to be replaced(r0), word to replace(r1), word to delete(d0). |
| Source | File selected from the computer. Changes made to the file in the GUI window’s text area. Local English dictionary from the program’s local folder. Word to find, word to be replaced, word to replace and word to delete from the GUI input window. |
| Outputs | NewFile-an edited file. |
| Destination | The selected file’s directory. |
| Action | NewFile is null if no file is open. NewFile is null if a file is open but is only used to “Find” a word. NewFile is the saved file c0 if a file is open and the “Save” button is clicked. Any action between “open” and “save”, including editing the file directly and using the “replace” or “delete” button to modify the file constitute c0. |
| Requires | A .txt or rtf file. |
| Precondition | At least a .txt or .rtf file to be opened. |
| Postcondition | None. |
| Side effects | None. |

**Graphical model (graph 1):**

**Operation**

Help

Do: Display help document

GUI Window

File is open

Edit

Do: Show edit menu items

Append file content

File open ok

File

Do: Show File menu items

Display result

File open failed

Open

Do: Open a .txt or .rtf file

Find

Do: Find a word

Replace

Do: Replace a word

Delete

Do: Delete a word

Save

Do: Save to opened file

File is open

Exit

Do: Exit the system

Close GUI window

Program terminated

**System Requirements Specification**

* 1. The system shall utilize a default graphical user interface (GUI) window which has a brief description of the system’s functionalities.
  2. The GUI window shall have a menu bar.
  3. The menu bar shall have three tabs called “File,” Edit” and “Help.”
  4. The “File” menu shall have a subtab called “Open” that is always enabled.
  5. If the “File” button is clicked, the system shall pop up a file selection window that allows the user to select to open a file.
  6. If the selected file is a .txt or .rtf file, the content of the file shall be displayed in the text area below the menu bar and shall be automatically adjusted to the boundaries of the GUI window.
  7. If the selected file is not a .txt or .rtf file, an error window shall pop up and return to the default GUI window.
  8. The “File” menu shall have a subtab called “Save” button that is initially disabled.
  9. If a text file is open, the system shall enable the “Save” button.
  10. If the “Save” button is clicked, the system shall save the current text content to the file’s original directory and output a message indicating file successfully saved.
  11. The “Edit” menu shall have a subtab called “Find” button that is initially disabled.
  12. If a file is open, the system shall enable the “Find” button.
  13. If the “Find” button is clicked, the system shall pop up a window that allows the user to input a word to find.
  14. The system shall first check the word against a local English library to make sure it is a valid English word.
  15. If the word is not an English word, the system shall pop up a window and ask the user if they wish to continue with that word.
  16. If the user chooses yes, the system shall search that word.
  17. If the user chooses no, the system shall return to the default GUI window.
  18. The “Edit” menu shall have a subtab called “Replace” button that is initially disabled.
  19. If a file is open, the system shall enable the “Replace” button.
  20. If the “Replace” button is clicked, the system shall pop up a window that allows the user to input the word to be replaced.
  21. The system shall first check the word against a local English library to make sure it is a valid English word.
  22. If the word is not an English word, the system shall pop up a window and ask the user if they wish to continue with that word.
  23. If the user chooses yes, the system shall search that word.
  24. If the user chooses no, the system shall return to the default GUI window.
  25. The system shall then pop up another window for the user to input a word to replace the previous word.
  26. The system shall replace a word only when it’s the same case as the word the user inputs.
  27. The system shall display the number of words successfully replaced as well as the number of words not replaced due to case sensitivity.
  28. The “Edit” menu shall have a subtab called “Delete” button that is initially disabled.
  29. If a file is open, the system shall enable the “Delete” button.
  30. If the “Delete” button is clicked, the system shall pop up a window that allows the user to input a word to delete.
  31. The system shall first check the word against a local English library to make sure it is a valid English word.
  32. If the word is not an English word, the system shall pop up a window and ask the user if they wish to continue with that word.
  33. If the user chooses yes, the system shall search that word.
  34. If the user chooses no, the system shall return to the GUI window.
  35. The system shall delete a word only when it’s the same case as the word the user inputs.
  36. The system shall display the number of words successfully deleted as well as the number of words not deleted due to case sensitivity in one window.
  37. The “Help” menu shall have a subtab called “Help” that is always enabled.
  38. If the “Help” button is clicked, the system shall pop up a window that explains in detail how the system works.
  39. The “File” menu shall have a subtab called “Exit” that is always enabled.
  40. If the Exit” button is clicked, the system shall correctly terminate the system. The system

shall also terminate when the GUI window is closed.

**Graphical model (graph 2):**

[Close GUI window]

Exits the program

Exit

Graphical User Interface window

Save

Save to opened file

File

[A file is open]

Display error message

[Is not .txt or .rtf file]

Open

Display file content in GUI window

[Is .txt or .rtf file]

[Find]

Local English dictionary

Find

[Replace]

[Is English]

Capture user input

Display result

Capture user input

Replace

Edit

[Delete]

[Yes]

[Is not English]

Delete

User chooses Continue or not

[No]

Display help document window

Help

**Platform Requirements Specification**

The software is an executable .jar file. It can be run on any operating system that has java runtime environment (JRE) installed. The system does not require a network connection to function.

**Architecture and Design Document**

**Introduction**

The system architecture consists of only one major component, which is the running process and a number of other sub-components. Specifically, the major component is the graphical user interface, and the sub-components are “File”, “Edit” and “Help” which are further consisted of lower sub-components, “Open”, “Save”, “Exit”, “Find”, “Replace”, “Delete” and “Help”. Therefore, this system architecture fits the patterns of the repository architecture. According to the description of the repository architecture given in our textbook, “all data in a system is managed in a central repository that is accessible to all system components. Components do not interact directly, only through the repository.” In this system, an opened text file in the GUI window is the central repository that manages all changes made to it and is accessible to all sub-components.

The choice of the repository architecture was made based on the user requirements, 2,3,4,5,6,7 and 8 in the Business Requirements Specifications document. User requirement 2 serves as the central repository by specifying that an opened text file must be opened in order for the subsequent requirements to function. Whereas user requirements 3,4,5,6,7 and 8 are all dependent on the existence of the central repository but independent of each other. Another non-functional requirement also specified by requirement 2 is that the text file must be either a .txt or .rtf file, which means that no other text format is acceptable by the system. However, this requirement does not directly affect to the choice of a system architecture. Some of the major characteristics of this system are performance, availability, and maintainability. Specifically, the performance is important because the system is a small piece of software that is intended for simple usages, which are finding, replacing and deleting a word from a file. Therefore, it should respond to its users’ input as quick as possible. Because of this characteristic, all components of the system are localized including a built-in English library for lookup. Availability is an important characteristic of this system also because it’s a localized system and should always be available to the users. Maintainability is also an important characteristic because the built-in English library needs to be updated regularly for any new English words and simply for the sake of code readability. This requires every component to be coded as modular as possible so that when one function is disabled for maintenance, other functions can still be used. However, there is a trade-off between availability and maintainability since when the one component is being maintained, it becomes unavailable to the users. One important underlying assumption in specifying the architecture is that most of the system functionalities can be used only after a file is open. In addition, the rest of the system functionalities are all independent of each other. These assumptions fit the description of the Repository architecture perfectly. There is no external system to my project’s system architecture.

**Architecture and Design Philosophy**

The key goal of my architecture is to take in an input and produce an output. This goal initially led me to the choice of the Pipe and Filter architecture since it does exactly what my goal describes. According to the textbook’s description of the Pipe and Filter architecture, “The processing of the data in a system is organized so that each processing component (filter) is discrete and carries out one type of data transformation. The data flows (as in a pipe) from one component to another for processing.” However, the textbook also states that, “Pipe and filter systems are best suited to batch processing systems and embedded systems where there is limited user interaction. Interactive systems are difficult to write using the pipe and filter model because of the need for a stream of data to be processed. While simple textual input and output can be modeled in this way, graphical user interfaces have more complex I/O formats and a control strategy that is based on events such as mouse clicks or menu selections. It is difficult to implement this as a sequential stream that conforms to the pipe and filter model.” First, my system requires a lot of user interactions in which the user needs to click on GUI window and input keyword. Secondly, it uses GUI to facilitate those user interactions. These two criteria of my system violate the latter description of the Pipe and Filter architecture given by the textbook. My next best choice is then the Repository architecture. In the textbook, it states that “This model [Repository architecture] is therefore suited to applications in which data is generated by one component and used by another.” This architecture also perfectly describes the goal of my system, which is to take in an input and produce an output and storing any changes made in between. In addition, the architecture has a central repository and several components that are dependent on it. This description also fits my system in which the GUI window and an opened file together serve as the central repository and the rest of the components are dependent on them and share their data. Since this is a stand-alone and small-scale system, it is only intended to serve one user at a time, therefore its user capacity and system load cannot be extended. However, in terms of upgrading the system, additional functionalities may be added to the system later easily because of the modularized architecture.

**Architectural Views**

Logical View:

The logical view of the text editor system is comprised of the 2 main components: User Interface and System Objects.

The User Interface component contains classes for each of the buttons that the actors use to interact with the system. Boundary classes exist to support opening a file, saving a file, exiting the system, finding a word, replacing a word, deleting a word, and showing the help document.

The System Objects Package includes entity classes and boundary classes for the interface.

Sub-system objects

OpenFile()

ReadFile()

SaveFile()

ExitSystem()

User interface

TextEditorGUI()

Sub-system objects

FindWord()

ReplaceWord()

DeleteWord()

WordValidityCheck()

System objects

FileMenu()

EditMenu()

HelpMenu()

Sub-system objects

HelpDocument()

Process View:

The Process Model illustrates the text editor classes organized as executable processes. Processes exist to support opening a file, finding a word, replacing a word, deleting a word, looking up the help document and exiting the software.

<<Process>>

GUI window

myGUI()

<<Process>>

FileMenu

FileMenu()

<<Process>>

SaveFile

SaveFile()

HelpDocument

//Display help document()

<<Process>>

OpenFile

OpenFile()

<<Process>>

EditMenu

EditMenu()

<<Process>>

Exit

//Exit software()

<<Process>>

DeleteWord

DeleteWord()

<<Process>>

ReplaceWord

ReplaceWord()

<<Process>>

FindWord

FindWord()

<<Process>>

WordValidityCheck

WordValidityCheck()

Development View:

This section is organized by physical network configuration; each such configuration is illustrated by a deployment diagram, followed by a mapping of processes to each processor.

<<Layer>>

GUI window

<<Layer>>

HelpDocument

<<Layer>>

Exit

<<Layer>>

FileMenu

<<Layer>>

OpenFile

<<Layer>>

EditMenu

<<Layer>>

SaveFile

<<Layer>>

FindWord, ReplaceWord, DeleteWord

<<Layer>>

WordValidityCheck

Physical View:

All components of the system are run on the user’s computer alone.

User’s PC

.jar executable

<<Process>>

GUI window

myGUI()

<<Process>>

FileMenu

FileMenu()

<<Process>>

SaveFile

SaveFile()

HelpDocument

//Display help document()

<<Process>>

OpenFile

OpenFile()

<<Process>>

EditMenu

EditMenu()

<<Process>>

Exit

//Exit software()

<<Process>>

DeleteWord

DeleteWord()

<<Process>>

ReplaceWord

ReplaceWord()

<<Process>>

FindWord

FindWord()

<<Process>>

WordValidityCheck

WordValidityCheck()

**Design Models**

Context Model of text editor:

<<System>>

Text Editor

<<Sub-system>>

GUI window

<<Sub-system>>

Help Menu

<<Sub-system>>

Edit Menu

<<Sub-system>>

File Menu

<<Sub-system>>

Find Word

<<Sub-system>>

Open File

<<Sub-system>>

Replace Word

<<Sub-system>>

Save File

<<Sub-system>>

Delete Word

<<Sub-system>>

Exit

Interaction Model of text editor:

Sequence diagram:

Text Editor GUI



Delete Word

Replace File

Find Word

Save File

Change content

Open File

OpenFile(File)

PrintToGUI(String)

File opened

PrintToGUI(String)

SaveFile(GUI,File)

Find Word(File)

ReplaceFile(GUI, File)

PrintToGUI(String)

DeleteWord(GUI, File)

PrintToGUI(String)

Exit

Help Document

Structural model of text editor:

Class diagram:

1 1

Output file

Input file

1 1..\*

1..\* 1

Input file

Change content

Query file

1

1..\*

1..\*

1..\*

Save file

Behavior Model of Text Editor:

Event-driven model:

**Operation**

Help

Do: Display help document

GUI Window

File is open

Edit

Do: Show edit menu items

Append file content

File open ok

File

Do: Show File menu items

Display result

File open failed

Open

Do: Open a .txt or .rtf file

Find

Do: Find a word

Replace

Do: Replace a word

Delete

Do: Delete a word

Save

Do: Save to opened file

File is open

Exit

Do: Exit the system

Close GUI window

Program terminated

Major data structure critical to the success of the system is hashing in the WordValidityCheck class for storing all the dictionary words into the hash set for look up.

Any system having java runtime environment (JRE) installed can run the system. As such there is no requirement for initialization files, configuration files or changes in system settings necessary.

**Testing Document**

**Unit Testing**

|  |  |
| --- | --- |
| Class | Method |
| 1. Project1 | main(String args[])  This is the driver method that starts the system by calling the TextEditorGUI class and passes a fixed string to it, no testing necessary. |
| 1. TextEditorGUI | TextEditorGUI(String s)  This is the constructor that sets up the parameters for the GUI window, no testing necessary. |
| createMenu()  This is a setter method that sets up the parameters for the Jbuttons, no testing necessary. |
| buttonEnable()  This is essentially a setter method that sets certain buttons to enable, no testing necessary. |
| setTextHelper(String text)  This is a setter method that sets the content of the JTextArea, no testing necessary. |
| 1. MenuHandler | MenuHandler(TextEditorGUI myGUI)  This is the constructor method, no testing necessary. |
| actionPerformed(ActionEvent event)  This method only receives one among the fixed events which cannot be changed by the user. All events are handled in this method, so no test necessary. |
| 1. FileMenu | FileMenu(TextEditorGUI myGUI)  This is the constructor, no testing necessary. |
| fileProcess(String choice)  This method only receives one among the fixed choices which cannot be changed by the user. All choices are handled in this method, so no testing necessary. |
| 1. TextFileOpener | TextFileOpener(TextEditorGUI myGUI)  This is the constructor, no testing necessary. |
| openFileHelper()   1. Test case 1: Input: .txt file. Output: “file open successful.”   Test case 2: Input: .rtf file. Output: “file open successful.”   1. Test case 3: Input: .doc file. Output: “This is not a .txt or .rtf file.”   Test case 4: Input: zip file. Output: “This is not a .txt or .rtf  File.” |
| 1. TextFileReader | TextFileReader(TextEditorGUI myGUI, File myFile)  This is the constructor, it calls the fileReaderHelper method, no testing necessary. |
| fileReaderHelper()   1. Receives the File object from its caller class, stores it in a string. 2. Catches IOException for BufferedReader. |
| 1. TextFileSaver | TextFileSaver(TextEditorGUI myGUI, File myFile)  This is the constructor, it calls the fileSaverHelper method, no testing necessary. |
| fileSaverHelper()   1. Writes the content in the JTextArea to the original File object. Output: “File successfully saved.” 2. Catches IOException for fileWriter. |
| 1. TextFileWriter | TextFileWriter(TextEditorGUI myGUI, int wordCount, String newWord, String oldWord, String choice)  This is the constructor, it calls the fileWriterHelper method, no testing necessary. |
| fileWriterHelper()   1. Input1: tempText: “This is a test string for testing purpose.”   Input2: oldWord: “test”  Input3: newWord: “experiment”  Output: newText: “This is a experiment string for testing purpose.”   1. No error handling necessary. |
| displayResult(int sucessfulReplacement, String choice)   1. Test case 1:   Input1: sucessfulReplacement = 0  Input2: choice = “Replace”  Input3: wordcount = 10  Output: "Number of ‘test’ successfully replaced with 'experiment’ is: 0  "Number of ‘test’ failed to be replaced because of case sensitivity is: 10  Test case 2:  Input1: sucessfulReplacement = 5  Input2: choice = “Replace”  Input3: wordcount = 5  Output: "Number of ‘test’ successfully replaced with 'experiment’ is: 5  "Number of ‘test’ failed to be replaced because of case sensitivity is: 5  Test case 3:  Input1: sucessfulReplacement = 10  Input2: choice = “Delete”  Input3: wordcount = 10  Output: "Number of ‘test’ successfully deleted is: 10   1. No error handling necessary. |
| 1. EditMenu | EditMenu(TextEditorGUI myGUI, File myFile, String myChoice)  This is the constructor, it calls the tabClicked method, no testing necessary. |
| tabClicked()  This method only calls one of the fixed choices which cannot be changed by the user. All choices are handled in this method, so no testing necessary. |
| 1. WordFind | WordFind(TextEditorGUI)  This is the constructor, it calls the finderHelper method, no testing necessary. |
| finderHelper()   1. Test case 1:   Input1: keyWord = test  Input2: myGUI JTextArea content: “This is a test string”  Output: “Number of ‘test’ in the file is: 1.”  Test case 2:  Input1: keyWord = null  Input2: myGUI JTextArea content: “This is a test string”  Output: JOptionPane window closes.  Test case 3:  Input1: keyWord = sssaaazzc  Input2: myGUI JTextArea content: “This is a test string”  Output: “This is not an English word; Do you wish to continue anyway?”  Test case 4:  Input1: keyWord = test  Input2: myGUI JTextArea content: “This is just a string”  Output: “Number of ‘test’ in the file is: 0.”   1. No error handling necessary. |
| 1. WordCount | WordCount(TextEditorGUI myGUI)  This is the constructor, no testing necessary. |
| wordCountFinder(String keyWord)   1. Test case 1:   Input1: myGUI JTextArea content: “This is a test string”  Input2: keyWord = test  Output: “Word count is 1”  Test case 2:  Input1: myGUI JTextArea content: “This is a TEST string”  Input2: keyWord = test  Output: “Word count is 1”  Test case 3:  Input1: myGUI JTextArea content: “This is a TEST string for test”  Input2: keyWord = test  Output: “Word count is 2”  Test case 4:  Input1: myGUI JTextArea content: “This is just a string”  Input2: keyWord = test  Output: “Word count is 0”   1. No error handling necessary. |
| 1. WordReplace | WordReplace(TextEditorGUI myGUI)  This is the constructor, it calls the replaceHelper method, no testing necessary. |
| replaceHelper()   1. Test case 1:   Input1: myGUI JTextArea content: “This is a test string”  Input2: oldWord = test  Input3: newWord= experiment  Output: Calls the TextFileWriter class.  Test case 2:  Input1: myGUI JTextArea content: “This is a test string”  Input2: oldWord = null  Output: JOptionPane window closes.  Test case 3:  Input1: myGUI JTextArea content: “This is a test string”  Input2: oldWord = test  Input3: newWord= null  Output: JOptionPane window closes.  Test case 4:  Input1: oldWord = sssaaazzc  Input2: myGUI JTextArea content: “This is a test string”  Output: “This is not an English word; Do you wish to continue anyway?”  Input3: Yes  Output: “There is no ‘sssaaazzc in the file.”  Input3: No  Output: JOptionPane window closes  Test case 5:  Input1: oldWord = test  Input2: myGUI JTextArea content: “This is just a string”  Output: “There is no ‘test’ in the file.”   1. Catches IOException for TextFileWriter. |
| 1. WordDelete | WordDelete(TextEditorGUI myGUI) This is the constructor, it calls the deleteHelper method, no testing necessary. |
| deleteHelper()   1. Test case 1:   Input1: myGUI JTextArea content: “This is a test string”  Input2: keyWord = test  Output: Calls the TextFileWriter class.  Test case 2:  Input1: myGUI JTextArea content: “This is a test string”  Input2: oldWord = null  Output: JOptionPane window closes.  Test case 3:  Input1: myGUI JTextArea content: “This is a test string”  Input2: oldWord = test  Input3: newWord= null  Output: JOptionPane window closes.  Test case 4:  Input1: oldWord = sssaaazzc  Input2: myGUI JTextArea content: “This is a test string”  Output: “This is not an English word; Do you wish to continue anyway?”  Input3: Yes  Output: “There is no ‘sssaaazzc in the file.”  Input3: No  Output: JOptionPane window closes  Test case 5:  Input1: oldWord = test  Input2: myGUI JTextArea content: “This is just a string”  Output: “There is no ‘test’ in the file.”   1. Catches IOException for TextFileWriter. |
| 1. WordValidityCheck | WordValidityCheck(String keyWord)  This is the constructor, no testing necessary. |
| validityCheckHelper()   1. Test case 1:   Input1: String str = test  Output: Return true  Test case 2:  Input1: String str= aascvxc  Output: "The word you entered is not a Engish word, contiune anyway?”  Input2: Yes  Output: Return true  Input3: No  Output: Return false   1. Catches IOException for InputStream InputStreamReader and BufferReader. |
| 1. HelpDocument | HelpDocument()  This is the constructor, it calls the printMessage method, no testing necessary. |
| helpMessage()  This is a setter method that sets the help message, no testing necessary. |
| printMessage()  This message prints the content of the helpMessage(), no testing necessary. |

**Component**

|  |  |  |
| --- | --- | --- |
| Classes involved | Methods involved | Interactions |
| 1. Project1 2. TextEditorGUI | 1. Main 2. TextEditorGUI constructor | 1. main method calls the TextEditorGUI method and passes a string to it.   Input: String s  Output: String s in a GUI window.   1. No error handling necessary since the string is fixed. |
| 1. TextEditorGUI | 1. TextEditorGUI constructor 2. createMenu | 1. TextEditorGUI calls the createMenu method to create the buttons.   Input: None  Output: A GUI window with buttons.   1. No error handling necessary since it just creates all the buttons. |
| 1. TextEditorGUI 2. MenuHandler | 1. createMenu 2. actionPerformed | 1. createMenu associates each button name with its action in the actionPerformed method.   Input: None  Output: None   1. No error handling necessary since all button names are fixed and all button actions are handled. |
| 1. MenuHandler 2. TextEditorGUI | 1. actionPerformed 2. buttonEnable | 1. actionperformed calls the buttonEnable method to enable certain buttons if a file is open   Input: None  Output: None   1. No error handling necessary since it is essentially a setter method. |
| 1. TextFileOpener 2. TextFileReader | 1. openFileHelper 2. TextFileReader | 1. openFileHelper allows the user to choose a file from their computer.   Input1: .txt file  Output: calls the TextFileReader with the file  Input2: .rtf file  Output: calls the TextFileReader with the file  Input3: .doc file  Output: “This is not a .txt or .rtf file."   1. No error handling necessary. |
| 1. WordFind 2. WordValidityCheck 3. WordCount | 1. finderHelper 2. validityCheckHelper 3. wordCountFinder | 1. finderHelper allows the user to input a word to find.   Input1: keyWord = test  Input2: myGUI JTextArea = “This is a test string”  Output1: calls validityCheckHelper with the keyWord  Output2: calls wordCountFinder with parameters keyWord and myGUI JTextArea content.   1. No error handling necessary. |
| 1. WordReplace 2. WordValidityCheck 3. WordCount | 1. replaceHelper 2. validityCheckHelper 3. wordCountFinder | 1. replaceHelper allows the user to input a word to find.   Input1: oldWord = test  Input2:newWord = experiment  Input3: myGUI JTextArea = “This is a test string”  Output1: calls validityCheckHelper with the oldWord and newWord  Output2: calls wordCountFinder with parameters keyWord and myGUI JTextArea content.   1. No error handling necessary. |
| 1. WordDelete 2. WordValidityCheck 3. WordCount | 1. deleteHelper 2. validityCheckHelper 3. wordCountFinder | 1. deleteHelper allows the user to input a word to find.   Input1: keyWord = test  Input2: myGUI JTextArea = “This is a test string”  Output1: calls validityCheckHelper with the keyWord  Output2: calls wordCountFinder with parameters keyWord and myGUI JTextArea content.   1. No error handling necessary. |
| 1. TextFileSaver 2. TextFileWriter | 1. fileSaverHelper 2. fileWriterHelper | 1. fileSaverHelper calls the fileWriterHelper method to save the content in the GUI JTextArea to the file.   Input: myGUI JTextArea = “This is a text string”  Output: file = “This is a text string”   1. Catches IOException for FileWriter. |

**System**

|  |  |
| --- | --- |
| Functional Requirement | Test |
| Requirement 1.1  Requirement 1.2  Requirement 1.3 | No need to test since these three requirements act like setters.  Project1 and TextEditorGUI classes create a GUI window and buttons “File,” Edit” and “Help,” and also display a brief message that describes what the system does. |
| Requirement 2.1 | No need to test since it just creates a “Open” button that is always enabled. |
| Requirement 2.2 | No need to test since it just creates a GUI file selection window that allows the user to open a file. |
| Requirement 2.3  Requirement 2.4 | Test case 1:  Input: a .txt file  Output: File content displayed in the text area |
| Test case 2:  Input: a .rtf file  Output: File content displayed in the text area |
| Test case 3:  Input: a .doc file  Output: Error message window “This is not a .txt or .rtf file.” |
| Requirement 3.1 | No need to test since it just creates a “Save” button that is initially disabled. |
| Requirement 3.2 | Test case 1:  Input: a .txt file  Output: File content displayed in the text area, “Save” button enabled. |
| Test case 2:  Input: a .rtf file  Output: File content displayed in the text area, “Save” button enabled |
| Test case 3:  Input: a .doc file  Output: Error message window showing “This is not a .txt or .rtf file.”  “Save” button disabled. |
| Requirement 3.3 | Test case1:  Input: 1. A file is open. 2. “Save” is clicked.  Output: File is saved to its originally destination. |
| Requirement 4.1 | No need to test since it just creates a “Find” button that is initially disabled. |
| Requirement 4.2 | Test case 1:  Input: a .txt file  Output: File content displayed in the text area, “Find” button enabled. |
| Test case 2:  Input: a .rtf file  Output: File content displayed in the text area, “Find” button enabled |
| Test case 3:  Input: a .doc file  Output: Error message window “This is not a .txt or .rtf file.”  “Find” button disabled. |
| Requirement 4.3 | Test case1:  Input: 1. A file is open. 2. “Find” is clicked.  Output: A GUI window that says, “Which word would you like to find?” and allows user input popped up. |
| Requirement 4.4  Requirement 4.5 | Test case1:  Input: 1. File string “This is a test file.” 2. String “test”  Output: “The number of ‘test’ in the file is 1” |
| Test case2:  Input: 1. File string “This is a test file.” 2. String “anhzzag”  Output: Message window showing “ This is not a English word, continue anyway?” |
| Requirement 4.6  Requirement 4.7 | Test case1:  Input: Yes  Output: “The number of “anhzzag” in the file is 0 |
| Test case 2:  Input: No  Output: //Returns to the default GUI window. |
| Requirement 5.1 | No need to test since it just creates a “Replace” button that is initially disabled. |
| Requirement 5.2 | Test case 1:  Input: a .txt file  Output: File content displayed in the text area, “Replace” button enabled. |
| Test case 2:  Input: a .rtf file  Output: File content displayed in the text area, “Replace” button enabled |
| Test case 3:  Input: a .doc file  Output: Error message window “This is not a .txt or .rtf file.”  “Replace” button disabled. |
| Requirement 5.3 | Input: 1. A file is open. 2. “Replace” is clicked.  Output: A GUI window that says, “Which word would you like to replace?” and allows user input popped up. |
| Requirement 5.4  Requirement 5.5 | Test case1:  Input: 1. File string “This is a test file.” 2. String “test”  Output: “Which word would you like to replace ‘test’ with” |
| Test case2:  Input: 1. File string “This is a test file.” 2. String “anhzzag”  Output: Message window showing: “ This is not a English word, continue anyway?” |
| Requirement 5.6  Requirement 5.7 | Test case1:  Input: Yes  Output: Message window showing: “Which word would you like to replace ‘anhzzag’ with?” |
| Test case 2:  Input: No  Output: //Returns to the default GUI window. |
| Requirement 5.8  Requirement 5.9 | Test case :  String “experimental”  Output: “Number of ‘test’ successfully replaced with ‘experimental’ is 1.” |
| Requirement 5.10 | Test case:  Input: 1. File string “This is a test file.” 2. String “TEST”  Output: “Number of ‘test’ successfully replaced with ‘experimental’ is 0. Number of ‘test’ that failed to be deleted because of case sensitivity is 1.” |
| Requirement 6.1 | No need to test since it just creates a “Delete” button that is initially disabled. |
| Requirement 6.2 | Test case 1:  Input: a .txt file  Output: File content displayed in the text area, “Delete” button enabled. |
| Test case 2:  Input: a .rtf file  Output: File content displayed in the text area, “Delete” button enabled |
| Test case 3:  Input: a .doc file  Output: Error message window “This is not a .txt or .rtf file.”  “Delete” button disabled. |
| Requirement 6.3 | Test case1:  Input: 1. A file is open. 2. “Delete” is clicked.  Output: A GUI window that says, “Which word would you like to delete?” and allows user input popped up. |
| Requirement 6.4  Requirement 6.5 | Test case1:  Input: 1. File string “This is a test file.” 2. String “test”  Output: “The number of ‘test’ deleted is 1.” |
| Test case2:  Input: 1. File string “This is a test file.” 2. String “anhzzag”  Output: Message window showing “ This is not a English word, continue anyway?” |
| Requirement 6.6 | Test case1:  Input: Yes  Output: “There is no “anhzzag” in the file.” |
| Requirement 6.7 | Test case 2:  Input: No  Output: //Returns to the default GUI window. |
| Requirement 6.8  Requirement 6.9 | Test case1:  Input: 1. File string “This is a test file.” 2. String “TEST”  Output: “The number of ‘test’ deleted is 0. The number of ‘test’ that failed to be deleted because of case sensitivity is 1.” |
| Requirement 7.1  Requirement 7.2 | No need to test since it just creates a “Help” button that is always enabled and displays a fixed message. |
| Requirement 8.1 | No need to test since it just creates a “Exit” button that is always enabled |
| Requirement 8.2 | Test case:  Input “Exit” button clicked  Output: System terminated. |

**Output of the executable software**

Default GUI window:

A screenshot of a cell phone

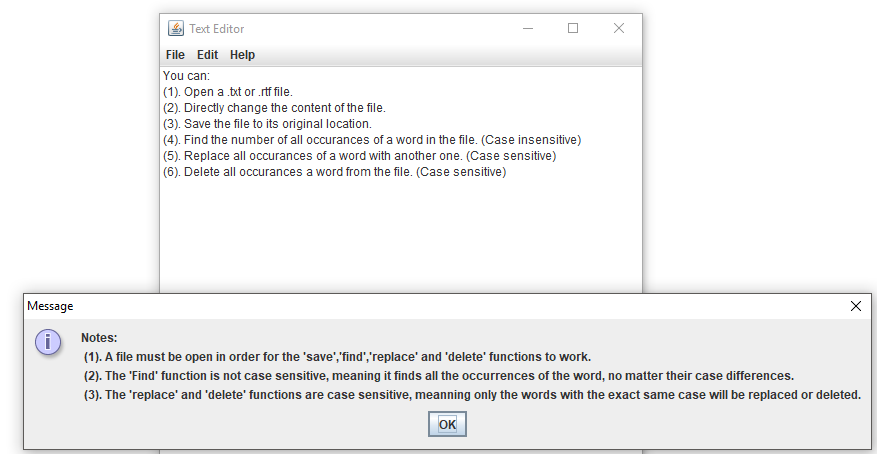
Description automatically generated**A screenshot of a cell phone

Description automatically generated**A screenshot of a cell phone

Description automatically generated**A screenshot of a cell phone

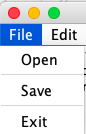
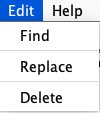
Description automatically generated**

Help document:

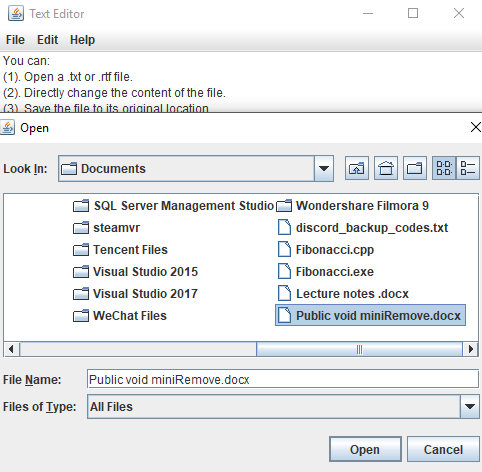


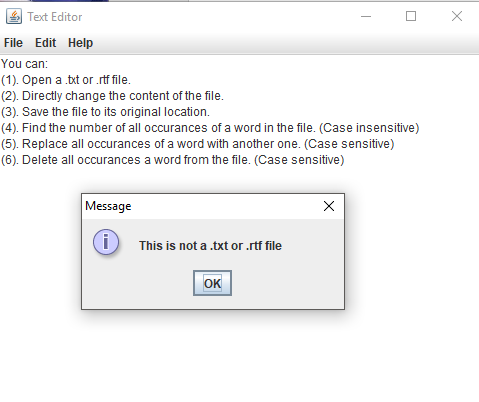
Open file successful:

A screenshot of a cell phone

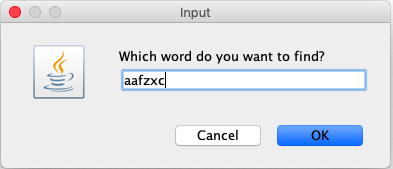
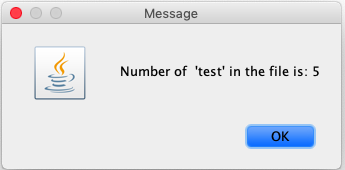
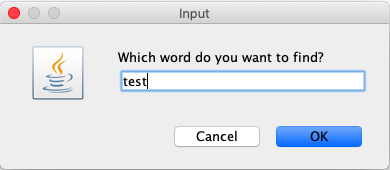
Description automatically generated  

Open file failed:



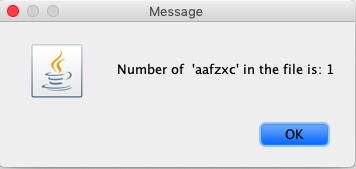


“Find” output:

A screenshot of a cell phone

Description automatically generated

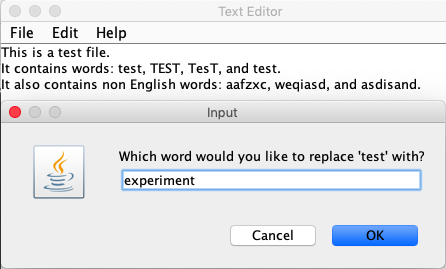




“Replace” output:

A screenshot of a cell phone

Description automatically generated



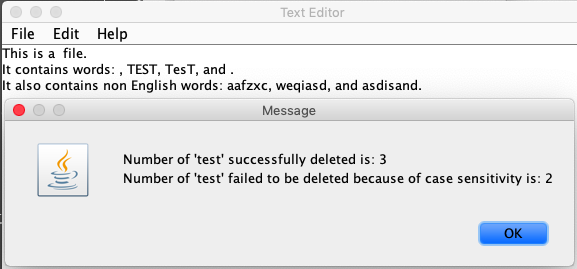
A screenshot of a cell phone

Description automatically generated

“Delete” output:

A screenshot of a cell phone

Description automatically generated



“Save” output:

“Save” output:

A screenshot of a cell phone

Description automatically generated

**Project Post-Mortem**

**What was learned:**

One of the things that I learned when developing my software was that it’s difficult to stick to the waterfall model throughout the course of the development cycle because inevitably you want to add new features when you are making your own software. Consequently, although the software development was supposed to follow the waterfall model, I had to go back to the requirements and design documents a few times to make new changes.

Another thing that I learned, programming-wise, was that you need some special tweaks in order to read .doc or .docx format files. Originally, I wanted my software to be able to read from those two file formats as well. Eventually I decided not to make that happen for now because those two formats essentially encrypted the text information, and in order to read the actual text information you need to decrypt them first.

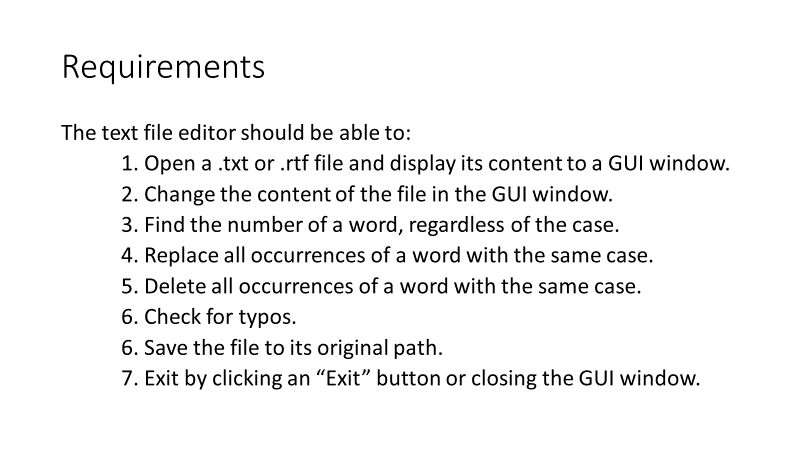
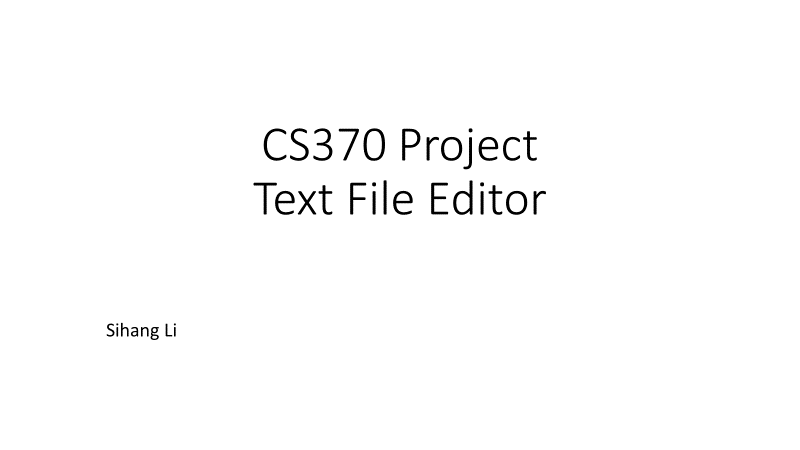
**What worked well:**

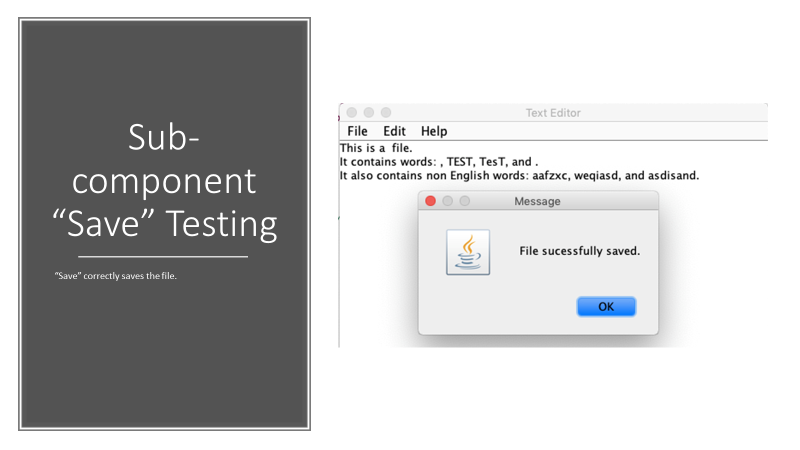
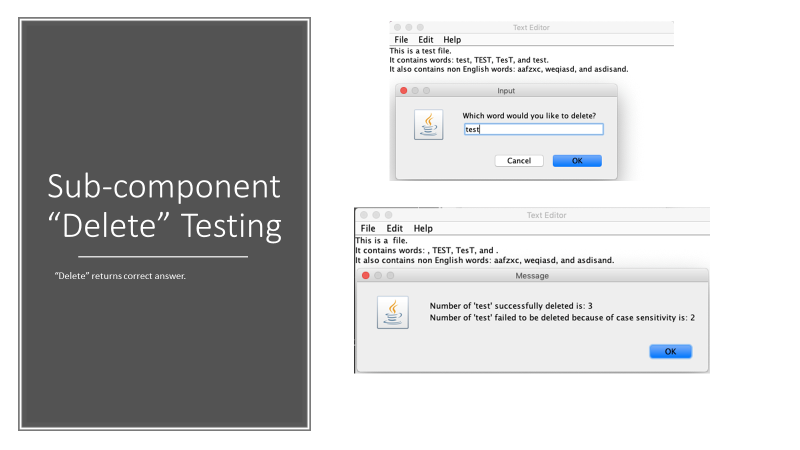
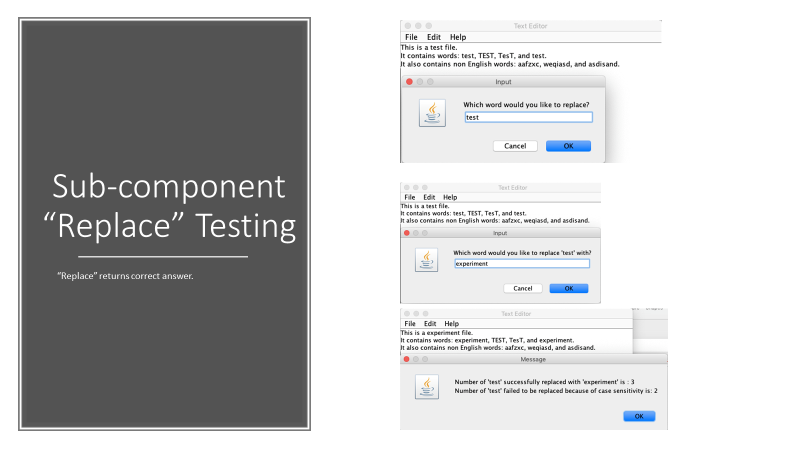
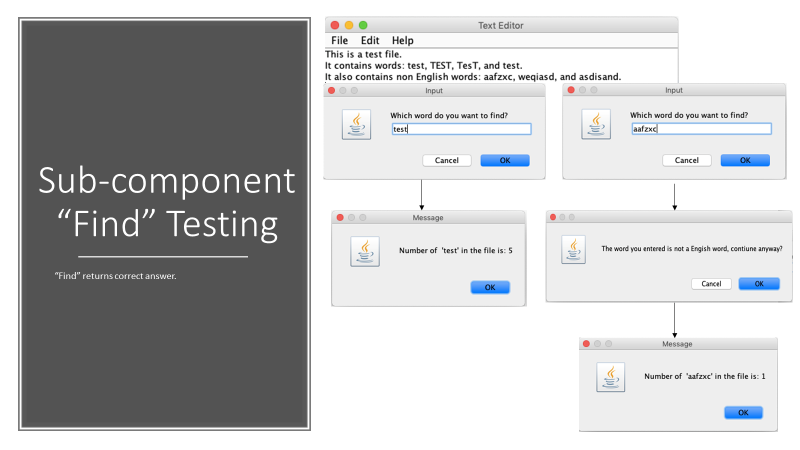
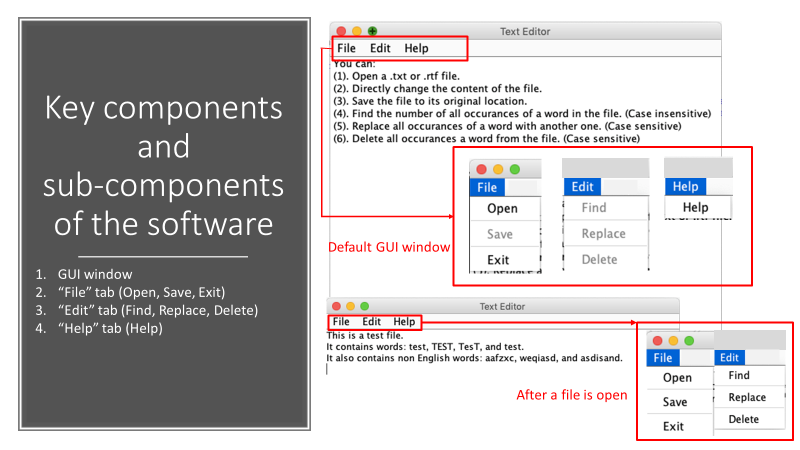
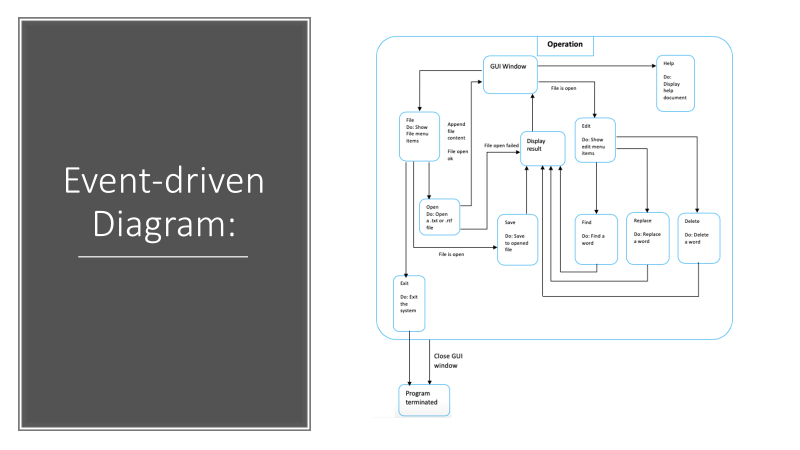
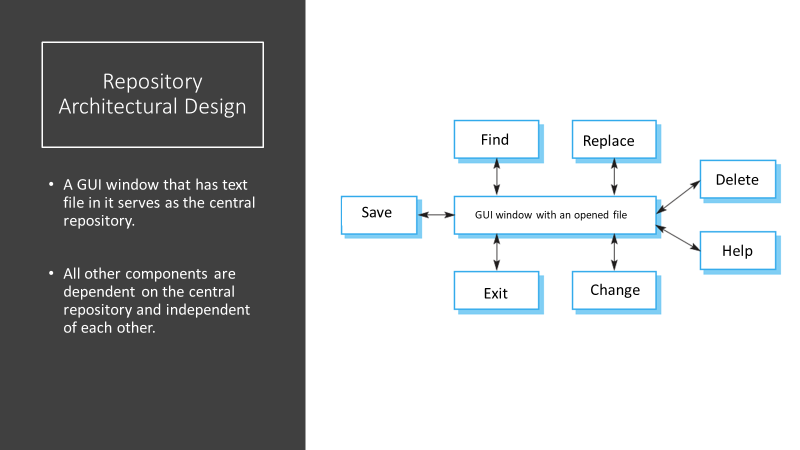
The software met all the requirements. All functionalities work correctly.

**What did not work well:**

Originally, I was going to use an online English dictionary for the word validity checking, such that my software can directly access the Internet and look up the word the user enters. It will consume less storage compared to having a local English dictionary, and have a promptly updated English word library. However, then I realized that in the case where the user doesn’t have Internet access, this functionality cannot used. Then I thought about using the English dictionary that’s built into every computer. Although this English library might not be updated promptly, it already there. However, this didn’t work as well because I was not able to get the English dictionary directory on my Macbook air. Eventually I settled for just putting a downloaded English Dictionary into the source code directory.

**Presentation slides:**





**Instruction on how to use my software:**

The software requires a JDE to work. Copy and paste each class into any JDE or IDE such as Eclipse and run from class “Project1”. Make sure to put the “Dictionary.txt” file in the same directory as the rest of the source code. The software itself has a brief instruction on how to use it in the default window.