Change request log CR #1

1. Concept Location

Step #	Description	Rationale
1	*We ran the system*	
2	*We interacted with the system.*	*To get familiar with some of the features of the system, and identify the graphical elements we had to change.*
3	*We searched for "offset" using the regular expression feature of the IDE's search tool.*	*Because we identified that character offset was already implemented in the system.*
4	*There were too many files matching this word and became extremely tedious to find out the required implementation.*	*Dropped this idea as it felt it was too generalized.*
5	*We searched for "StatusBar" keyword, we found many files but the file named "StatusBar.java" caught our attention.*	*As it is named as StatusBar, it felt that its implementation would be within that class as the name suggests.*
6	*In this class, there was a method "UpdateCaretStatus", where in one of the "if" block the caret offset was being added to the status bar to display.*	*We wanted to figure out where from exactly the caret position was being coming up on screen from the code.*
7	*Thus, one file was located. That is, the file where we could add the word offsets to the status bar. But, we also needed to figure out from where is the data actually coming.*	*To perform any word count or word offset operation, the primary necessity is the source data.*
8	*We observed a variable named "bufferLength", it was getting its data from a method called "getTextContext"*	*The character count is also implemented in the system, so there must be a way to read the whole text from the window and "getTextContext" method also suggests to do the same.*
9	*The "getTextContext" method was implemented in jEditBuffer class. So navigated to that class, we observed that there was a variable "contentMgr" which holds the complete data of the text editor.*	*The data was being returned to the buffer using the object "contentMgr" of type "ContentManager".*
10	*Thus, we understood the source of the data of the text editor and also the class from where we can set values on the status bar.*	*Thus, location of classes was complete.*

Time spent (in minutes): 100

Example:
Classes and methods inspected:

```
        org/gjt/sp/jedit/gui/StatusBar.java
                public void updateCaretStatus()
        org/gjt/sp/jedit/buffer/JEditBuffer.java
                public CharSequence getTextContext()
        org/gjt/sp/jedit/ContentManager.java
               public int getLength()
               public int getLineCount()
              public int getLineOffset(int offset)
              public int getLineStartOffset(int line)
             public int getLineEndOffset(int line)
```

2. Impact Analysis

Step #	Description	Rationale
1	*Utilized regular expression search to find occurrences of "offset" within the codebase.*	*Conducted to identify existing implementations or references to the concept of "offset" within the system, which is crucial for assessing the potential impact of introducing changes related to word offsets in the status bar.*
2	*Abandoned the search for "offset" due to its generality and the difficulty of finding the required implementation.*	*Decision made to avoid investing further resources in a generalized search that yielded numerous results, indicating potential complexities and uncertainties in pinpointing specific areas for impact analysis.*
3	*Identified a file named "StatusBar.java" where the word "offsets" could be added to the "status bar".*	*Essential step to determine the location for implementing changes for accurate impact analysis and subsequent modifications to the status bar display.*
4	*Realized the need to find the source of the data.*	*This is a crucial step, as we can perform further steps, i.e. word count and calculating word offset only if we have the source data.*
4	*Investigated the "UpdateCaretStatus" method within "StatusBar.java" to understand caret position handling.*	*Impact analysis focused on understanding existing mechanisms related to caret position, crucial for determining the feasibility and potential implications of integrating word offsets into the status bar functionality without disrupting existing features.*
5	*Discovered the "bufferLength" variable retrieving data from "getTextContext" method.*	*Identification of data source vital for impact analysis, as it enables tracing the flow of relevant information within the system*
6	*Explored the "getTextContext" method in the jEditBuffer class, which was managing text editor content via "contentMgr" variable.*	*Understanding the source of data and its management provides insights into the system's architecture.*

Time spent (in minutes): 40

Example:
Classes and methods inspected:

```
        org/gjt/sp/jedit/gui/StatusBar.java
                public void updateCaretStatus()
        org/gjt/sp/jedit/buffer/JEditBuffer.java
                public CharSequence getTextContext()
        org/gjt/sp/jedit/ContentManager.java
               public int getLength()
```

3. Actualization

Step #	Description	Rationale
1	*We created a new method in "JEditBuffer" class named "getTextContext" which has a return type of CharSequence.*	*The creation of the "getTextContext" method allows for the encapsulation of text retrieval logic within the "JEditBuffer" class, enhancing modularity and abstraction. This approach promotes better code organization and maintainability, facilitating future modifications and ensuring consistent data retrieval throughout the system.*
2	*The above method was being called from the "StatusBar.java" class. The whole content was successfully being retrieved to the StatusBar class.*	*By invoking the "getTextContext" method from the "StatusBar.java" class, the system gains access to the complete content, enabling comprehensive analysis and manipulation. This approach centralizes data retrieval, promoting consistency and reducing redundancy across different parts of the codebase.*
3	*Now, that we have the total content with us, we calculated the total number of words in the document. We added a "for loop" to count the number of words by splitting the words based on the space in between them*	*The implementation of a "for loop" to count words based on space delimiters enables accurate word counting within the document. This approach provides a straightforward and efficient method for calculating word counts, ensuring reliable data analysis and presentation in the status bar.*
4	*Then the task of the calculate the word offset based on the caret position. There was a existing variable named "caretPosition" in the method "updateCaretStatus".*	*Leveraging the "caretPosition" variable within the "updateCaretStatus" method streamlines the process of calculating word offsets based on the caret position. By utilizing existing variables, the code maintains simplicity and minimizes the introduction of unnecessary complexity or dependencies.*
5	*Making use of this, we wrote another "for loop", ran the loop until the "caretPosition" and once more calculated the words until the caret position.*	*The addition of a "for loop" to calculate words until the caret position enhances the functionality of the system by providing real-time updates on word offsets. This approach ensures accurate word offset calculations, enhancing the usability and effectiveness of the status bar feature.*

Time spent (in minutes): 60

Example:
Classes and methods inspected:

```
        org/gjt/sp/jedit/gui/StatusBar.java
                public void updateCaretStatus()
        org/gjt/sp/jedit/buffer/JEditBuffer.java
                public CharSequence getTextContext()
        org/gjt/sp/jedit/ContentManager.java
               public int getLength()
```

4. Validation

Step #	Description	Rationale
1	*Test case defined:* *Inputs: Empty screen with no words and no spaces* *Expected output: 0/0*	*This is the regular expected behavior.* *The test passed.*
2	*Test case defined:* *Inputs: Screen with single word and cursor at the beginning of the word* *Expected output: 0/1*	*This is the regular expected behavior.* *The test passed.*
3	*Test case defined:* *Inputs: Screen with single word and cursor at the end of the word* *Expected output: 1/1*	*This is the regular expected behavior.* *The test passed.*
4	*Test case defined:* *Inputs: Screen with 5 words and cursor at the end of the 5th word* *Expected output: 5/5*	*This is the regular expected behavior.* *The test passed.*
5	*Test case defined:* *Inputs: Empty screen with no words but 10 spaces* *Expected output: 0/0*	*This is an exceptional behavior (when the screen is completely empty but the cursor is being moved on the screen)* *The test passed.*
6	*Test case defined:* *Inputs: Screen with 5 words and multiple spaces in between them and cursor at the end of the 5th word* *Expected output: 5/5*	*This is an exceptional behavior (when there is multiple spaces in between the words, to check if our logic is able to identify moving from a empty space to a word)* *The test passed.*
7	*Test case defined:* *Inputs: Screen with 5 words and moving the cursor across these 5 words and checked if the offset is being updated accurately.* *Expected output: all the values are being changed as expected when the cursor moves on the screen*	*This is the regular expected behavior.* *The test passed.*

Time spent (in minutes): 40

5. Summary of the change request

Phase	Time (minutes)	No. of classes inspected	No. of classes changed	No. of methods inspected	No. of methods changes
Concept location	60	3	2	5	2
Impact Analysis	40	3	2	3	2
Prefactoring					
Actualization	60	3	2	2	2
Postfactoring					
Verification	40	3	2	2	2
Total	200	11	8	12	8

6. Conclusions

For this change, concept location took quiet sometime as there were many files which had the term "offset" and "status bar". We used the IntelliJ IDE for the concept location, impact analysis and actualization. Testing was done by manually coming up with test cases. This also took quiet some time and effort to come up with edge cases.