Change request log

1. Team

Specify the team members working on this change request.

Driver: Ivan Huerta-Bernal

Navigator: Michael Petrey

2. Change Request

6.1 Change request #1: Addition of current word offset of the caret in the status bar area.

3. Concept Location

Use the table below to describe each step you follow when performing concept location for this change request. In your description, include the following information when appropriate:

- IDE Features used (e.g., searching tool, dependency navigator, debugging, etc.)
- Queries used when searching
- System executions and input to the system
- Interactions with the system (e.g., pages visited)
- Classes visited
- The first class found to be changed (this is when concept location ends)

When there is a major decision/step in the process, include its rationale, i.e., why that decision/step was taken.

Step #	Description	Rationale
1	*With the project loaded in the IDE we compiled and ran jEdit to see how the feature to be changed worked.*	*To observe what affects does the user input have on the system. We observed that the bottom lefthand corner counts the caret position both via coordinate position (row, col) and (currentChar / totalChars).*
2	*Looked at the source directory names to hypothesis which one to look in first. We figured that directory "org" was a good choice.*	*With a quick look through the folders, org contained jedit which seemed like a good place to start.*
3	*Upon expanding the "org" folder we found two other folders "gjt" and "jedit" we expanded both using Eclipse's Package Explorer View. To view its subdirectories*	*Because expanding 2 subdirectories is not too hard to do and diving deeper into the directories and looking at the subdirectory names will give us a better understanding of how the system is organized.*
4	*We opted to expand **jedit** first.*	*Because we weren't sure why a subdirectory would be named **gjt. Jedit** made more sense.*
5	*Upon expanding **jedit** we saw other directories like "**core**", "**io**", "**keymap**". We quickly expanded those using Eclipse's Package Explorer view.*	*We needed to dive deeper to see what classes we could find inside the **org/jedit** subdirectories.*
6	*We found that many of classes inside **org/jedit** dit not pertained to what we were looking for we took to looking inside the **org/gjt/sp/jedit** directory.*	*Because we had found nothing that pertained to a user interface of any kind.*
7	*This folder seemed more promising because we saw class files and other directories with names like **gui**, **menu**, **search**, **GUIUtilities.java**. We opted to expand **org/gjt/sp/jedit/gui***	*Because it pertained to what we were looking to do. Make a change to the GUI.*
8	*We opened StatusBar.java and quickly saw that we had made it to the correct place.*	*The brief description at the top of the source file which describes the information provided by it helped us confirm that we had made it to the correct class. **StatusBar.java** has to be modified. Furthermore, using*

		*Eclipse's Outline we were able to quickly see the methods that were implemented in the class. More specifically the method **updateCaretStatus().***
9	*We jumped to the analyze the implementation of the **updateCaretStatus()** method. This looked promising so we concluded the concept location step.*	*Because this method looked like a good staring point. We saw that this is where the (caretPosition/bufferLength) tracker was implemented.*

Time spent (in minutes): 65

4. Impact Analysis

Use the table below to describe each step you follow when performing impact analysis for this change request. Include as many details as possible, including why classes are visited or why they are discarded from the estimated impact set.

Do not take the impact analysis of your changes lightly. Remember that any small change in the code could lead to large changes in the behavior of the system. Follow the impact analysis process covered in the class. Describe in detail how you followed this process in the change request log. Provide details on how and why you finished the impact analysis process.

Step #	Description	Rationale
1	*We looked at the return type of **updateCaretStatus()***	*We needed to see if there was the possibility that this method returns somethings another classes/object rely on.*
2	*We analyzed the methods that were being called inside **updateCaretStatus()***	*We needed to analyze how the called methods affect **updateCaretStatus().** Analyze why they were being called.*
3	*First, we found that view.getBuffer() was called. Then we used Ctrl+F to see what **view** had to do with anything. We found that **StatusBar's** constructor takes a view object as an argument, so Wo opened **View.java.***	*We need to analyze what **View.java** is responsible for.*
4	*Inside the **View** class we used Ctrl+F to look for the method **getBuffer**().*	*Being that **getBuffer()** was what was being called in **StatusBar.java** we needed to analyze it further.*
5	*Once we found **getBuffer()** we found that it returned a Buffer, but that buffer came from an EditPane object. Seeing that we turned to looking for an EditPane class and found it in **org/gjt/sp/jedit.** We opened the class file to analyze it.*	*We needed to see what the EditPane class was responsible for. We needed to get a good mental picture of how the different classes worked together.*
6	*We then saw that an instance of a **JEditTextArea** object was(curiously) created using **view.getTextArea(),** the text area from the view. We looked for and found the **JEditText** class, so we opened it for further analysis.*	*We needed to see where the methods **getCaretPosition()** and **getCaretLine()** were implemented. Mostly to further expand our knowledge about how the system was organized. (Being that they were only used as index variables in **StatusBar.java**)*
7	*We did not find much in **JEditText*** but we did see that it extended **TextArea.** We saw there was a **textarea** directory, so we expanded it and opened the **TextArea** class.	*To continue expanding our knowledge of the system.* *It is good that we did because inside the class we found several implementations of **getText()** which return String.*
8	*We figured that if we calculated the word counts similarly to how the character counts are implemented, we should be good to go.*	*We finished impact analysis by concluding that as long as we first tried to implement that change at this local level(in the **updateCaretStatus** method) and relying on functionality already written then impact would be minimal.*

Time spent (in minutes): 55

5. Actualization

Use the table below to describe each step you followed when changing the code. Include as many details as possible, including why classes/methods were modified, added, removed, renamed, etc.

Step #	**Description**	**Rationale**
1	*The first change we made was implement a method that counts to total number of words in the opened file.*	*This is a core component of the change request. We took this part on first because we saw that there were several calls to a buffer object in this method and from our impact analysis, we knew that the buffer object could return a String which we could use to return the total word count in the buffer.*
2	*From our impact analysis we knew that the local buffer object would return a string when calling **buffer.getText().** Our **getTotalWordCount()** would rely on getting the giant String on the screen, splitting if up using Java regex .split() method, adding the words to an array and lastly returning the size of the array.*	*We figured this was a simple enough idea. That if it did not work then we could quickly move past it to try something else.*
3	*We wrote the **getTotalWordCount(String lns)** method inside **StatusBar.java** to return an integer (the array size after all the words are added to an array). The we called it from inside **updateCaretStatus** by passing **buffer.getText()***	*We needed to see if **buffer.getText()** a String as expected, furthermore, that it would be a String that included everything in the currently opened buffer.*
4	*Next it was time to implement the word count at the caret position. We recalled from our impact analysis that there existed another **getText()** method that also returned a string and took start and end parameters.* *So we implemented **getWordCountAtCaret()** much like our first method but this time we passed to it **getText(0, caretPosition).*** *Where **caretPosition** was already defined for us as part of the implementation to get the character count.*	*We went about it this way because much of the functionality was already implemented for the **caretPosition/bufferLength** tracker originally implemented, and because it was the most straightforward and simple thing to try.*

Time spent (in minutes): 45

6. Validation

Use the table below to describe any validation activity (e.g., testing, code inspections, etc.) you performed for this change request. Include the description of each test case, the result (pass/fail) and its rationale.

Step #	**Description**	**Rationale**
1	*Test case defined: Testing a user created file.* *Inputs: We used the common nursery rhyme: Mary had a little lamb* *Expected output: When moving the caret around we should see the total word count stay the same and the word offset change.*	*The total word count is 5 and the word offset changes accordingly. This was to test a user made text file and the count performed as expected.*

2	*Test case defined: Testing the word updates while making changes.* *Inputs: We used the common nursery rhyme: Mary had a little lamb* *Expected output: While typing the phrase the total count should be increasing with each word typed.*	*The word count grew as the phrase was typed. This function performed as expected.*
3	*Test case defined: Testing a larger file.* *Inputs: Opening README.SRC.txt to test a larger word count.* *Expected output: There should be a high total word count and the caret word offset should update as you move through the file.*	*When the file is opened the word count shows 1326 and does not change as the caret traverses the file. The caret word offset changes in the expected manner as the caret traverses the file.*

Time spent (in minutes): 30

7. Timing

Summarize the time spent on each phase.

Phase Name	**Time (in minutes)**
Concept location	65
Impact Analysis	50
Actualization	45
Verification	30
Total	190

8. Reverse engineering

Create a UML sequence diagram (or more if needed) corresponding to the main object interactions affected by your change.

Create a partial UML class diagram of the classes visited while navigating through the code. Include the associations between classes (e.g., inheritance, aggregations, compositions, etc.), as well as the important fields and methods of each class that you learn about. The diagram may have disconnected components. Use the UML tool of your preference. When a significant fact about a class or method is learned, indicate it via annotations on the diagram. **For each change request, start with the diagram produced in the previous change request. For the first, you will start from scratch.**

9. Conclusions

Perform an analysis of the change requests and the change process. List the major challenges this change request posed.

List all the classes and methods you have changed.

The concept location and impact analysis took quite some time. This due to the fact that it was the first time looking at the system/code, so it took a long time to become familiar with the location of components/classes. Additionally, once we did recognize that StatusBar.java had to modified (more specifically the *updateCaretStatus* method) we observed that several calls were made to instantiate objects from somewhere else in the code base, mainly *JEditTextArea* and *Buffer* objects. Finding those classes in the codebase was straightforward but making sense of how and why they were implemented as they are took some time. Both are large classes that extend from other classes and call other classes. It took more time to dive through those and analyze how they were related and why they were being called inside *updatedCaretStatus.* The originally implemented *caretPostion/bufferLength* counter was of big help because the it was straightforward to append the information requested in the change request to the caretPosition/bufferLength and because a *caretPosition* variable was already created. Concept location helped us arrive at the class where the change should be made, but impact analysis helped up come up with the best way to go about implementing the change request.

Class changed

- *org/gjt/sp/jedit/gui/StatusBar.java*

    added

getTotalWordCount(String fileWords)
getWordCountAtCaret(String fileWords)

modified
updateCaretStatus()