Change request log

# Team

Team reNo

Matt Dragan and Ben Sattelberg

# Change Request

Change request 2.1

The left side of the status bar of jEdit reports: the line number containing the caret, the column position of the caret, the character offset of the caret from the beginning of the file, and the number of characters in the file (see Figure 2). You are requested to modify the status bar to show: the word offset of the caret from the beginning of the file and the number of words in the file. As a result of this change, the status bar for the example of Figure 2 should report: “2,6 (76/138886)(12/23731)”.

# 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.

**Make sure you time yourselves when going through this process and provide the total time spent below.**

The following is an example of a concept location process for the change request "Color student schedule":

|  |  |  |
| --- | --- | --- |
| **Step #** | **Description** | **Rationale** |
| **1** | *We searched the jEdit documentation for “word count.”* | *If there is a word count functionality that already exists, we do not want to replicate it unnecessarily.* |
| **2** | *We used Eclipse’s search tool to search for “word count.”* | *Since jEdit already implements a word count function, we want to use it if possible.* |
| **3** | *Of the results that existed, only one was in a Java file (the other was in the documentation files). We went to to JEditTextArea.java file.* | *We want to investigate the flow of the code, rather than the front facing documentation that we investigated in step 1.* |
| **4** | *We found that the “doWordCount()” function in the “JeditTextArea.java” file implements a word count.* | *We determined that a word count functionality was already implemented. We noted this for future use and investigated other potential changes.* |
| **5** | *We used Eclipse’s search tool to look for files named “StatusBar.java.”* | *In the change request, the location of the change was referred to as “status bar.” Ideally, the code would match the conceptual name of the object.* |
| **5** | *We inspected the class “StatusBar.java.”* | *We wanted to locate how the character and line numbers were currently displayed.* |
| **6** | *We found that the “updateCaretStatus()” function called the “TextArea.java” functions “getCaretPosition()” and “getCaretLine().”* | *We were interested in how the current information was displayed. We found that the text area keeps track of the location of the caret and its line position.* |
| **9** | *We found that the “updateCaretStatus()” function called the “Buffer.java” function “getLineCount()” and “getLength().”* | *The current implementation for display on the status bar uses these functions.* |
| **10** | *We investigated the “getLineCount” and “getLength()” functions in “Buffer.java.”* | *We found that the “Buffer.java” has LineManager and PositionManager objects it uses to keep track of the locations* |
| **11** | *We investigated the “LineManager.java” and “PositionManger.java” classes.* | *This were what was currently being used for tracking caret location.* |
| **12** | *We discussed possibilities for how to implement the word count:*   * *Create a “WordManager.java” class* * *Use a modified version of the “doWordCount()” function in “JEditTextArea.java”* | *We have two options for implementing the word count and we must decide which to use.* |
| **13** | *We decide to use a modified version of the “doWordCount()” function in “JeditTextArea.java”* | *Both options are appealing.*  *The implementation of the “WordManager.java” class would be complex and seems to potentially use Java structures we are not familiar with. It also would be a significant change for this change request.*  *The implementation of the modified “doWordCount()” will likely be slower, as the word count will be recomputed each update. However, it will be a simpler change. It also naturally fits the definition of a “word” that JEdit is currently using.*  *We decide to modify “doWordCount(),” but if it preforms excessively poorly we will reconsider.* |
| **14** | *We marked the class “StatusBar.java” as “located.”* | *We will need to modify this file to calculate word counts.* |

**Time spent (in minutes):** 152

# 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.

**Make sure you time yourselves when going through this process and provide the total time spent below.**

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 marked “JeditTextEditor.java” as to be changed.* | *The functionality for counting words exists in this class, but we need it to be modified slightly to be accessible.* |
| **1** | *We made a list of methods that call “updateCaretStatus()” or use the “caretStatus” attribute of “StatusBar.java.”* | *To track the classes that could be impacted by the change.* |
| **2** | *We found no classes that used the “caretStatus” attribute beyond a JPanel that is entirely controlled by the “StatusBar.java” class. No external modifications need to be made based on this.* |  |
| **3** | *We inspected the class “EditPane.java.” It was discarded from the list of changes.* | *This class only calls “updateCaretStatus()” as the caret moves. No changes must be made here.* |
| **4** | *We inspected the class “View.java.” It was discarded from the list of changes.* | *This class only calls “updateCaretStatus()” as the caret moves. No changes must be made here.* |
| **5** | *We made a list of methods that are called by “doWordCount()” (it is protected, so nothing outside the class will call it).* | *To track the classes that could be impacted by the change.* |
| **6** | *We inspected the class GUIUtilities. It was discarded from the list of changes.* | *The “message(...)” method of this class is called by “doWordCount().” However, we are not planning on changing the functionality of “doWordCount(),” so it is unnecessary to change it.* |
| **7** | *We finished the impact analysis.* | *No other classes appeared to be impacted by the changes.* |

**Time spent (in minutes):** 20

# Prefactoring (optional)

Using the table below, describe each step you follow to prefactor the code. Include as many details as possible, including the refactoring operations used (e.g., move method, extract class, etc.) and classes/methods/fields that were modified, added, removed, renamed, etc.

**Make sure you time yourselves when going through this process and provide the total time spent below.**

|  |  |  |
| --- | --- | --- |
| **Step #** | **Description** | **Rationale** |
| **1** | *We extracted the word count functionality from “doWordCount()” in “JeditTextArea.java.” We used the refactoring “extract method” by hand.* | *The method “doWordCount()” originally counted the number of words and sent it to a display message. We need the word count without sending it to the display.* |
| **2** | *After the previous change, we ran the existing unit tests (although they seem to only exist for an unrelated part of the system) and also we ran the system. We used the edit menu’s word count display function.* | *We tested everything was working as before, after the refactoring.* |

**Time spent (in minutes):** 10

# 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.

**Make sure you time yourselves when going through this process and provide the total time spent below.**

|  |  |  |
| --- | --- | --- |
| **Step #** | **Description** | **Rationale** |
| **1** | *We modified the “updateCaretStatus()” function in “StatusBar.java” to use the newly extracted word count functionality.* | *Added functionality.* |
| **2** | *We ran the existing test cases (again we don’t believe these test anything related to the changes).* | *To make sure everything works.* |
| **3** | *We performed functional testing.* | *To make sure everything works.* |
| **4** | *We committed and pushed our changes with git.* | *Just in case we need to revert our changes.* |

**Time spent (in minutes):** 5

# Postfactoring (optional)

Use the table below to describe each step you followed to postfactor the code. Include as many details as possible, including the refactoring operations used (e.g., move method, extract class, etc.) and classes/methods/fields that were modified, added, removed, renamed, etc.

**Make sure you time yourselves when going through this process and provide the total time spent below.**

|  |  |  |
| --- | --- | --- |
| **Step #** | **Description** | **Rationale** |
| **1** | *We did not do any postfactoring.* | *Changes made during actualization only modified a small section of the code that did not seem to require any changes.* |

**Time spent (in minutes):** 1

# 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.

**Make sure you time yourselves when going through this process and provide the total time spent below.**

|  |  |  |
| --- | --- | --- |
| **Step #** | **Description** | **Rationale** |
| **1** | *Code review of changes made* | *Ensure that changes make sense.* |
| **2** | *Test case defined:*  *Inputs: …*  *Expected output:...* | *This is an exceptional behavior (when values are larger than 100 characters)*  *The test passed.* |
| **3** | *Functional testing – verify that word count displays correctly* | *This displayed the expected behavior.* |
| **4** | *Functional testing – verify that the word count from the edit menu displays correctly still* | *This displayed the expected behavior.* |

**Time spent (in minutes):** 20

# Timing

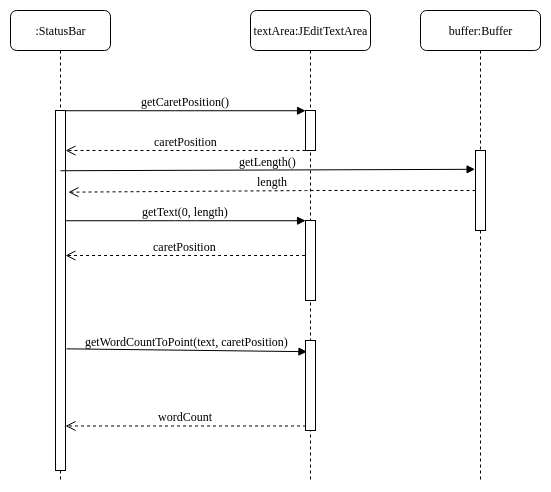
Summarize the time spent on each phase.

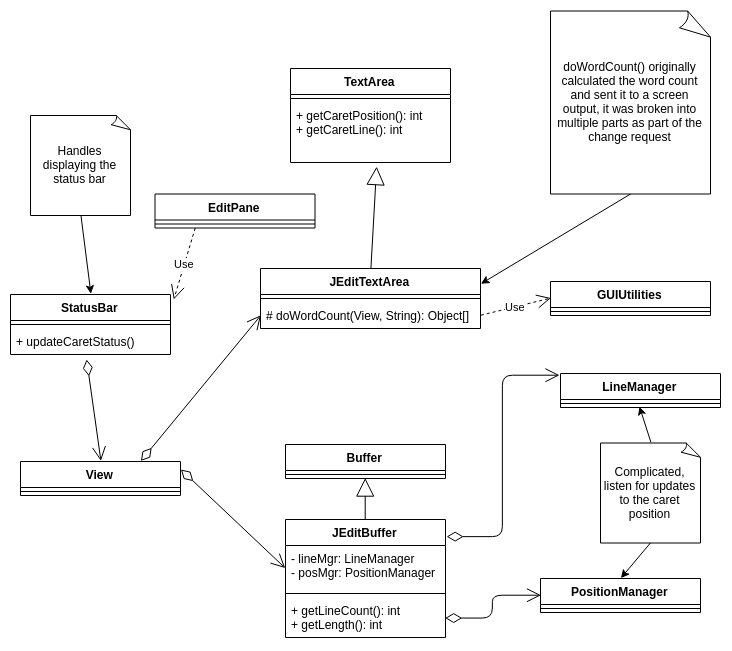
|  |  |
| --- | --- |
| **Phase Name** | **Time (in minutes)** |
| Concept location | 152 |
| Impact Analysis | 20 |
| Prefactoring | 10 |
| Actualization | 5 |
| Postfactoring | 1 |
| Verification | 20 |
| **Total** | 208 |

# 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.**

****

****

# Conclusions

Provide a set of conclusions about the change request and the change process. List the major challenges this change request posed.

List all the classes and methods you have changed.

*For this change, concept location was relatively difficult because we are not experienced with Java or this style of project. However, the project was simple enough we were able to determine how it worked. Concept location and impact analysis were largely done using Eclipse’s search functionality. Prefactoring and actualization were done by hand in Eclipse. For this change, we had significant difficulty with Eclipse not playing nicely with our machines (it seems to have minor, but annoying, bugs that appear when using Linux systems). We also had significant difficulty understanding decisions that were made by the JEdit developers and interpreting Java code. Constructing UML sequence and class diagrams was also difficult, as we have not had enough practice creating those to understand precisely how they should be structured. We also don’t have any tools for UML generation that we are familiar with, so the creation in draw.io was somewhat difficult and tedious.*

*Matt and Ben pair programmed the modifications to the program.*

*Ben created the change request and the UML diagrams.*

*Classes and methods changed:*

* *org/gjs/sp/jedit/gui/StatusBar*
  + *void updateCaretStatus()*
* *org/gjt/sp/jedit/textarea/JeditTextArea*
  + *void doWordCount(View, String)*
  + *Object[] getWordCountToPoint(String, int)*
  + *Object[] getWordCount(String)*