Change request log

# Team

reNo – Matt Dragan and Ben Sattelberg.

# Change Request

Change request 2.2

In the File » Recent Files main menu of jEdit, the text box on top of the recent files list allows to highlight recent files names that match with a given string (see Figure 2). The string in the text box should match all the files that contain it anywhere in their name. However, the highlight works only when the string matches the beginning of a file name. You are requested to modify this feature so that the highlight occurs for the cases when the string is contained anywhere in the file name.

# 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** | *Ran jEdit* | *Verify that the requested change doesn’t already exist* |
| **2** | *Use Eclipse search tool to search for “recentfiles”* | *Since looking at recent files already exists it is likely that the change would occur where the recent file details are handled.* |
| **3** | *Of the search results we chose to check out the RecentFilesProvider.java* | *This result seemed the most promising for finding where recent file logic is handled.* |
| **4** | *Use Eclipse local search tool to search for “search” ,“bar” , or “searchbar”* | *Search potentially useful queries to assist with concept location.* |
| **5** | *Inspect the update() method for concept location* | *This is likely the function responsible for updating the state of the recent files method.* |
| **6** | *Inspect keyReleased() function* | *Indicates how the code responds to an input key.* |
| **7** | *Mark the keyReleased() function passed as “located”* | *We confirmed that this function is the function that needs to be modified.* |

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

# 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 analyzed what what functions are called by the keyReleased() function.* | *We want to track functions and classes that could be impacted by making changes to this function.* |
| **2** | *We determined that keyReleased() is passed as a parameter to the addKeyListener() function of the JTextField class which is a built in java method* | *The impact of a change to keyReleased() occurs in a built in java method that we are unable to change.* |
| **3** | *We analyzed what methods call the keyReleased() function.* | *We want to tack functions and methods that could be impacted by making changes to this function.* |
| **4** | *We determined that the functions called by keyReleased() are parts of classes that are not part of jEdit.* | *We don’t need to worry about changing these functions because we are unable to access this code.* |

**Time spent (in minutes):** 27

# 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** | *N/A* | *We determined that prefactoring was not necessary for this change.* |

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

# 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 added a “\*” to the beginning of the regular expression to allow for any number of leading characters before a pattern.* | *This allows for any number of characters to exist before the pattern when pattern matching.* |
| **2** | *Ran the built in unit tests. (We aren’t sure that these unit tests test for these changes)* | *Ensure that all existing unit tests pass* |

**Time spent (in minutes):** 2

# 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** | *N/A* | *We determined that postfactoring is not necessary for chis change* |

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

# 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* | *Reviewed the code to make sure that the existing changes make sense* |
| **2** | *Ran existing unit tests. (Although, we aren’t sure that these test the particular part of the program we are working with.)* | *Make sure that existing unit tests still pass* |
| **3** | *Functional testing* | *Ran the system to verify that the change behaves as expected* |

**Time spent (in minutes):** 4

# Timing

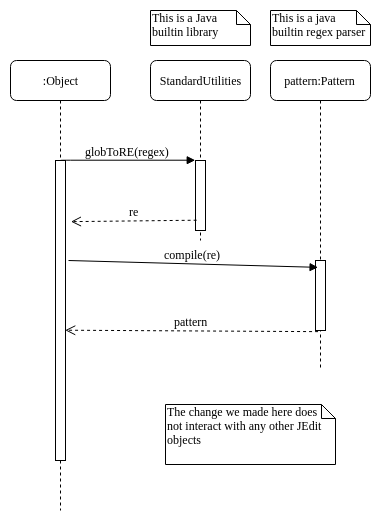
Summarize the time spent on each phase.

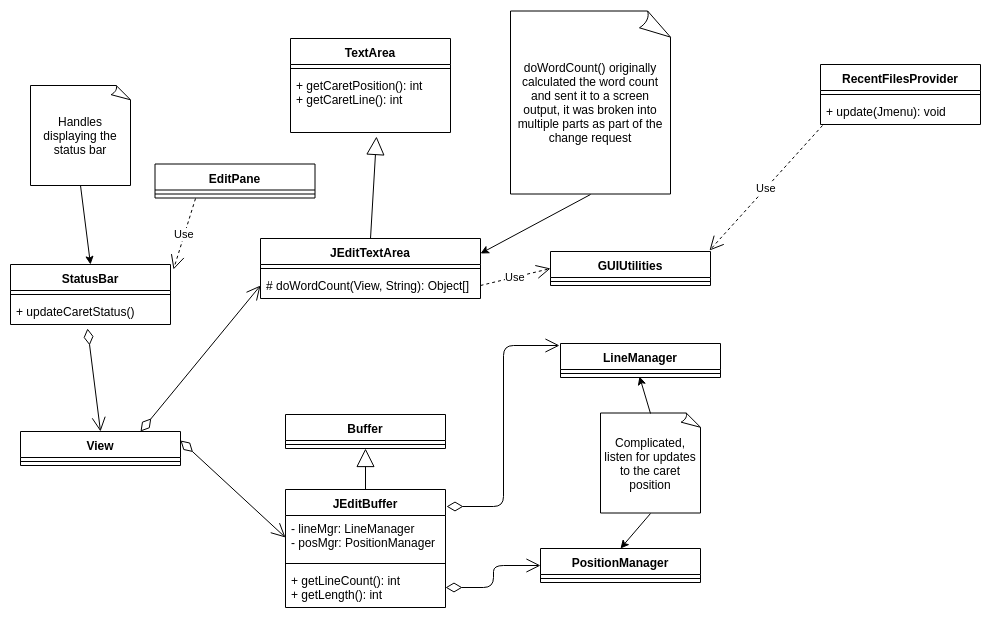
|  |  |
| --- | --- |
| **Phase Name** | **Time (in minutes)** |
| Concept location | 10 |
| Impact Analysis | 27 |
| Prefactoring | N/A |
| Actualization | 2 |
| Postfactoring | N/A |
| Verification | 4 |
| **Total** | 43 |

# 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 example:

For this change, concept location was relatively easy because the naming scheme of the system made it easy to locate with a simple search. Searching for recent files led us to the exact module that required change. Impact analysis was relatively simple because it doesn’t appear that any jEdit modules call the function that needed to be changed and the function itself doesn’t call any of the other jEdit modules. We just had to determine that the functions called in the function we changed are not part of jEdit and therefore shouldn’t change the functionality. Actualization was simple with an understanding of how regular expressions work and required minimal changes to the code. The UML diagrams were relatively easy because the change was small will low impact.

Matt wrote this log with proofreading from Ben. Ben created diagrams.

Ben and Matt pair programmed the modification to this program.

Classes and methods changed:

* org/gjt/sp/jedit/menu/RecentFilesProvider.java/RecentFilesProvider
  + void update(Jmenu)
    - void keyReleased(KeyEvent)