Change request log

1. Team
Driver: Matthew Starks

Navigator: Ivan Huerta-Bernal

2. Change Request
6.2 Change request #2

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.

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

Step #	Description	Rationale
1	*We compiled and ran the project.*	*To take a closer look at how the Recent Files search feature worked.*
2	*This time around, we had a better mental model about how the project was structured. We immediately used Eclipse's Package Explorer view and went to the* ***org/gjt/sp/jedit*** *directory*	*At this point we had a better idea of the structure of the project knew that it likely that this change would be implemented in some file inside the* ***org/gjt/sp/jedit*** *directory. We understood that this directory housed a lot of the core functionality of the project.*
3	*Once here we expanded the* ***gui*** *folder. And visually searched (using Eclipse's Package Explorer view) for something relating to "menu"*	*We believe this to be a GUI change so logically it was the first folder to search in.*
4	*Then we expanded the* ***menu*** *directory. We expanded that directory using Eclipse's Package Explorer.*	*We found nothing relating to "menu" in the* ***gui*** *directory.*
5	*The opened the* ***RecentsFilesProvider.java*** *class.*	*Because it relates to something we are looking for. The "recent files" search functionality.*
6	*We skimmed through the code using the main editor window.*	*Because this file is not too large, and it only has one method.*
7	*We concluded the concept location step.*	*Because we found a Listener that seemed to implement a String construction using regex and found the keywords "recent" and "menuItems" which are thing we are looking for as part of this change.*

Time spent (in minutes): 40

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 details 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 the code carefully and re-ran the project.*	*We need to see whether in the case that nothing is typed in the search bar items were highlighted or grayed out. During the re-run we confirmed that the empty string case highlighted every recent item. This further helped us confirm that all the original feature is doing is modify the way the regex string is built. See lines 100 - 120*
2	*Below where the regex string was being built we saw a set of **try/catch** statements that seem to rely on the declared **filter** variable. We made a note to keep that as part of our modification.*	*Our initial aim was to modify as little code as possible and thought it best not to modify anything more than the way the regex was being built if possible.*
3	*Our impact analysis concluded.*	*After we concluded that the impact of our change would be minimal being that the method where the String build was being implemented returned **void** and that all we needed to do was modify the way the regex string was being built. From the look of it, all we needed to do was "sandwich" the **typedText*** in between two asterisks(the regex wildcard).

Time spent (in minutes): 40

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	*To implement the change first we moved the **regex** String outside the **if(filter)** statement and initialized it to an empty String.*	*This because we needed to instantiate the variable before the **if** statement but not using the **typedText** right away like originally implemented.*
2	*We opted to remove the if statement containing, **if(!typedText.contains("*"))..***	*Because we observed that the **if(!typedText.contains("*"))..** was actually the portion of the code responsible for the Recent Files search only taking into account the initial part of the string typed by the user. Because all it did was add an asterisk to the end of the string.*
3	*We opted to comment out this entire section and simply concatenate an asterisk, then the typedText, then another asterisk.*	*Our quick analysis of this section of code pointed to this being okay. We also were not 100% sure of being in the correct location. It is possible that this whole class belonged to a different feature.*
4	*The first implementation did not work, so we tried a different version of the implementation.*	*We had to modify our original implementation because the project failed to compile. Turns out the **filter** variable in the section of code we commented out was required.*
3	*We added the **if(filter)..** section back in. We called our previously defined **regex** variable inside this if statement and concatenated an*	*Being that our first attempt failed and our original plan was to sandwich the **typedText** in between two asterisks. This to ensure that the functionality of "highlight occurs for the*

	*asterisk, then the **typedText,** followed by another asterisk.*	*cases when the string is contained anywhere in the file name" was accomplished.*
4	*We saved, and ran the project and proceeded to validate the change.*	*It seemed like it was working.*

Time spent (in minutes): 35

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: First case is to test multiple ways of searching for the same file. Using README.SRC.txt* *Inputs: rea, DM, S, SR ,c.t , e, AdMe* *Expected output: Highlighted README.SRC.txt*	*The regular expected behavior occurred. The file was highlighted with each iteration of the search.* *The test passed.*
2	*Test case defined: Second case is to test multiple files.* *Inputs: Open all the text documents in the doc folder and search bits of their names.* *Expected output: The names that have portions that match the search text will be highlighted while the ones that don't are not.*	*The test yielded expected behavior with each like-file being highlighted while the others were not.*

Time spent (in minutes): 25

7. Timing

Summarize the time spent on each phase.

Phase Name	Time (in minutes)
Concept location	40
Impact Analysis	40
Actualization	35
Verification	25
Total	140

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

Implementing this change was relatively straightforward. Previous "explorations" through the code made us more confident about the decision we made as we proceeded through the concept location and impact analysis. This time around it was easier to select what directory to expand on next, or which class to analyze further. After we found the **RecentFilesProvider** class in the **menu** directory, and saw that the *update* method returned void all we had to do is carefully analyze the 15is lines that dealt with the building of the regex string that is passed to the pattern matcher. Although there were no blatant signs that we had indeed arrived at the right place, all the keywords surrounding the implementation(class name, variables, code) pointed to us being in the correct spot. We only were 100% sure that we were after we commented out the original code and implemented the simplest possible implementation of the change. Attempting to compile this first implementation resulted in an error due to use failing to see the functionality of the **filter** variable. We modified our implementation to use the **filter** variable, compiled and tested. The modification worked as expected.

Classes and methods changed:

- *org/gjt/sp/jedit/menu/RecentFilesProvider.java*
  - *void update(JMenu menu)*
    - *text.addKeyListener({new KeyAdapter() .... public void keyReleased(KeyEvent e)...... });*