Change request log

# Team

Specify the team members working on this change request.

Angela Chen & Uma Kothuri

# Change Request

Provide the id and description of the change request.

jedit-cr-4

In the File » Recent Files main menu of jEdit, the Clear Recent Files option on the bottom of the recent files allows to clear recent files (see Figure 4). Similarly, in the Utilities » Recent Directories main menu, there should be Clear Recent Directories option on the bottom of the recent directories allows to clear recent directories. You are requested to implement this feature to clear the recent directories list.

# 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 used IntelliJ’s “ctrl+shift+f” search tool to find the file RecentDirectoriesProvider.java using the keyword “recentdirectories”. | We determined “recent directories” to be the main concept for this change request. |
| 2 | We found a possibly related class RecentDirectoriesProvider within RecentDirectoriesProvider.java. | We determined RecentDirectoriesProvider class to be the class to be changed, since it is for the recent directories menu. Concept location is complete. |
| 3 |  |  |
| 4 |  |  |
| 5 |  |  |
| 6 |  |  |
| 7 |  |  |
| 8 |  |  |
| 9 |  |  |

**Time spent (in minutes):** 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 looked through the file where RecentDirectoriesProvider class was located to find any other classes it call on. | To find any classes that RecentDirectoriesProvider maybe depend on. |
| 2 | We found an object of interface type MouseListener created in the file and eliminated it from the impact set. | MouseListener’s definition was not related to concepts from the change request. |
| 3 | The MouseListener object is of type MouseAdapter, we found the MouseAdapter class and eliminated it from the impact set. | MouseAdapter’s definition was not related to concepts from the change request. |
| 4 | We found a HistoryModel object through more manual search in the RecentDirectoriesProvider class and found the HistoryModel.java file that defines the HistoryModel class. We eliminated it from the impact set. | HistoryModel’s definition was related to the history menu in the change request, but we determined that changes will not impact the history menu itself, so the class will not be impacted. |
| 5 | We looked through the RecentFilesProvider class and found the area of code related to its history recent files menu. | We are familiar with the class RecentFilesProvider through a previous change request and deduced that it should be related to the change request. |
| 6 | Using “ctrl+shift+f” with “clear recent files” and found a properties file jedit\_en.props that defines part of the code from step 5. We used static search with ctrl + f to find the corresponding block of code for recent directories instead of recent files. We will add this file to the impact set. | We determined that any classes impacted by the code from step 5 should be part of the impact set for the history menu to be implemented for RecentDirectoriesProvider as well. Although the file is not a class, it will be changed in actualization, so we have decided to add it to the impact set. |
| 7 | We have finished impact analysis. | We determined that we have used all necessary models to find any impacted classes. |

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

# 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 | We are basing the actualization from an area of code of RecentFilesProvider.java implementing its recent files history. | We determined RecentFilesProvider to have the option that needs to be implemented for RecentDirectoriesProvider. |
| 2 | We used the code but replaced BufferHistory with HistoryModel objects. | RecentDirectoriesProvider class uses HistoryModel instead of BufferHistory objects like RecentFilesProvider to keep its history. |
| 3 | We changed the jedit\_en.props property file to implement a label for the clear recent directories option based on the implementation for the label for the clear recent files option. | We determined RecentFilesProvider to have a similar label as the one that needs to be implemented for RecentDirectoriesProvider. |

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

# 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:  Inputs: multiple directories saved in history, press “clear recent directories” option  Expected output: history is cleared | This is the expected behavior after implementing the change request.  The test passed. |
| 2 | Test case defined:  Inputs: one directory saved in history, press “clear recent directories” option  Expected output: history is cleared | This is the expected behavior after implementing the change request.  The test passed. |
| 3 | Test case defined:  Inputs: no directories saved in history (history is empty), try to press “clear recent directories” option  Expected output: unable to press option (option is disabled) | This is the expected behavior after implementing the change request.  The test passed. |

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

# Timing

Summarize the time spent on each phase.

|  |  |
| --- | --- |
| Phase Name | Time (in minutes) |
| Concept location | 4 |
| Impact Analysis | 20 |
| Actualization | 5 |
| Verification | 2 |
| Total | 31 |

# 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

Perform and 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.

For example:

*For this change, concept location was relatively easy because the system is small and its architecture and code are not complicated. Concept location, impact analysis, actualization (and change propagation) was done using JRipples, which was very useful. Testing was performed using JUnit and Abbot. It took me long time to make Abbot work, mainly because I had to learn how to use the tool.*

*Classes and methods changed:*

* *org/gjt/sp/jedit/search/HyperSearchResults.java/HyperSearchResults*
  + *void foo(p:integer)*
* *org/gjt/sp/jedit/search/HyperSearchResults.java/HyperSearchFind*
* *etc.*

For this change, each phase of the change process was relatively easy because the change concepts were similar to the previous change request we did (jedit-cr-2), so we were already familiar with related classes and how they work. This made the concept location and actualization easier. We also used the same software (IntelliJ) as the previous change request for the change process, so the process involved no difficulties with the software as well.

Classes and methods changed:

* org/gjt/sp/jedit/menu/RecentDirectoriesProvider.java
  + void update(JMenu menu)
* org/jedit/localization/jedit\_en.props (not a class, but we changed the file)
  + Recent Directories menu (block of code that we changed)