Change request log

# Team

Brad and Nate

# Change Request

Change Request #1

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 1). 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 to learn how the recent file search currently works. |  |
| 2 | Searched for the uses of JMenu. | The recent files were in a JMenu so we thought it was a good start. |
| 3 | Found a JMenu in GUIUtilities so we debugged loadMenu() | First JMenu in the class |
| 4 | Debugging through loadMenu() led me to the EnhancedMenu class. |  |
| 5 | Debugged through EnhancedMenu to learn how items were added to the menu | EnhancedMenu seemed like a good name for a menu that is probably being used. |
| 6 | Noticed the RecentFileProvider was called when you went to the Recent Files menu item in the UI |  |
| 7 | Looked through the RecentFileProvider because it had a lot of events in it. | *ScheduleUI uses the method getBlock. However, we realized this class was not relevant for our change request, because it is rather related to the data layer instead to the GUI layer.* |
| 8 | Found a key pressed event that fired when we typed in the filter box. | *We were not sure if this method had to be changed, therefore we decided to debug.* |
| 9 | Debugged to find where it did a regex compare | *We confirmed this class had to be modified.* |

**Time spent (in minutes):** 30

# 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 | All the variable are local except for turning a file option to enabled or not so the impact is very low. | *To track the classes that could be impacted by the change.* |
| 2 | *We inspected the class GUIRenderer. Such class was marked as "to change" as well* | *We realized this class had to be changed because the method render uses the StudenGraph to get the properties of the schedule. Then, the new property, i.e., color, has to be registered in the map of properties.* |
| 3 | *...* |  |
| 4 | *The class SchedulePanel was discarded from the list of classes to change* | *Because the class deals with the rendering the panel instead of the table of the schedule* |

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

# 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.

|  |  |  |
| --- | --- | --- |
| Step # | Description | Rationale |
| 1 | *We converted the variable color into a field in class Schedule (method changeColor). We used the refactoring "extract field" from the IDE.* | *As many methods will access the color value, it is a good idea to have a field. This would reduce the number of arguments and parameters of the methods* |
| 2 | *After the previous change, we ran the unit tests corresponding to the class Schedule and also we ran the system. We went to the schedule screen.* | *We tested everything was working as before, after the refactoring.* |
| 3 | *We committed our changes with git.* | *Just in case we need to revert our changes.* |
| 4 | *...* |  |

**Time spent (in minutes):** 15

# 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 | Added a \* character to the beginning of the regex expression | This allows anything to be before the word as well as the already existing after wild card character. |
| 2 | *...* |  |
| 3 | *We created unit tests for the new class and also performed functional testing. We also ran the existing test cases.* | *To make sure everything works.* |
| 4 | *...* |  |

**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.

|  |  |  |
| --- | --- | --- |
| Step # | Description | Rationale |
| 1 | *We converted the variable color into a field in class Schedule (method changeColor). We used the refactoring "extract field" from the IDE.* | *As many methods will access the color value, it is a good idea to have a field. This would reduce the number of arguments and parameters of the methods* |
| 2 | *After the previous change, we ran the unit tests corresponding to the class Schedule and also we ran the system. We went to the schedule screen.* | *We tested everything was working as before, after the refactoring.* |
| 3 | *We committed and pushed our changes with git.* | *Just in case we need to revert our changes.* |
| 4 | *...* |  |

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

# 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 | Tried to find a file with the key word in the middle | The test passed. |
| 2 | Tried to find a file with the key word in the beginning to be sure we did not break previous functionality | Test passed |
| 3 |  |  |
| 4 | *...* |  |

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

# Timing

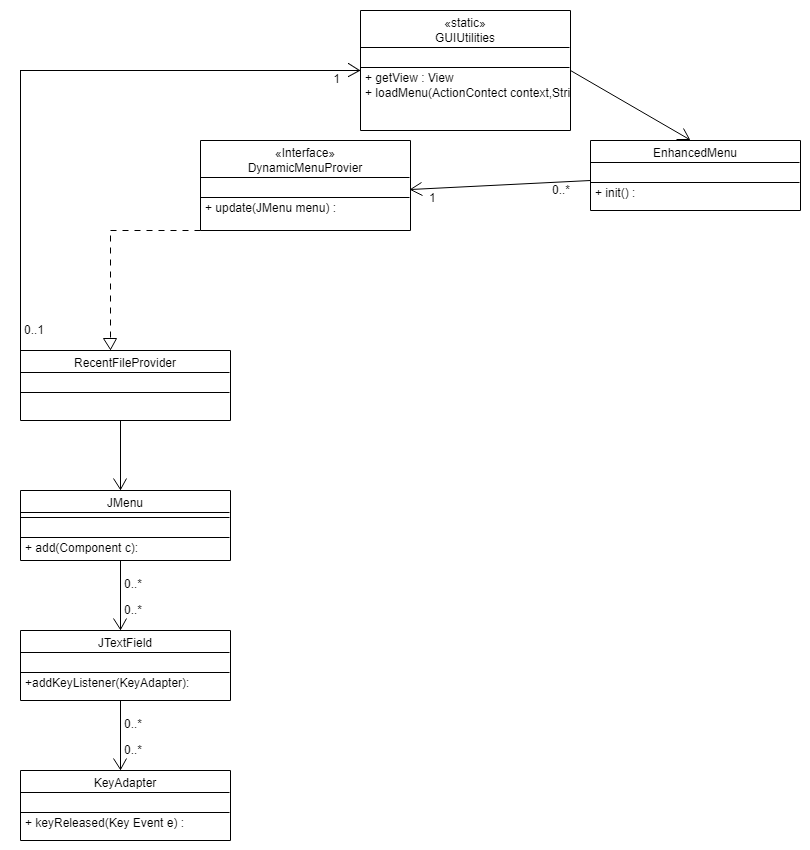
Summarize the time spent on each phase.

|  |  |
| --- | --- |
| Phase Name | Time (in minutes) |
| Concept location | 30 |
| Impact Analysis | 10 |
| Prefactoring | 0 |
| Actualization | 2 |
| Postfactoring | 0 |
| Verification | 5 |
| Total | 47 |

# 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:

This change request was easy because it was just changing a regex string to include another wild card

Classes and methods changed:

* org/gjt/sp/jedit/menu/RecentFileProvider.java
  + public void update(JMenu menu)
    - public void KeyReleased(KeyEvent e)