Change request log

# Team

Specify the team members.

# Change Request

Provide the id and description of the change request.

# 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 | Run the program to learn the users path of execution. | To learn where the change request was in the UI and how it originally worked. |
| 2 | Look through the code for the Utilities and Favorites options. | This could lead to the button pushed event that I think would fill the browser window. |
| 3 | Search for "provider" since the other menu item was recentfileprovider.java | We learned from request #1 that providers handle the menu options. |
| 4 | Debug through the FavoritesProvider class | See if anything in the class calls another useful class. |
| 5 | The Favorites seem to be stored in a FavoritesVFS collection so I look at all the uses of that | Not much uses the Favorites class directly but they do call the FavoritesVFS class which has a collection of Favorite objects. |
| 6 | We find a class called BrowserView that has the FavoritesVFS object in it so I will debug this class too | Just looking at a class that is related to the FavoritesVFS class |
| 7 | When Edit Favorites is clicked the BrowserView is created so we have found the correct class | This class was affected by a mouse click in the area we are concerned with. |
| 8 | Not much came up looking through the BrowserView class |  |
| 9 | We found a class called ListDirectoryBrowser Task that is called anytime there is an action on the table I want to change | Since the BrowserView class was the only one affected by mouse actions |
| 10 | After looking through the BrowserView class we found the GUI table VFSDirectoryEntryTable | Kept looking through the BrowserView execution path with the debugger. |
| 11 | After being stuck we decided to look through every method of the Favorites class. | We found that this is where the data to fill the VFSDirectoryEntryTable was. |
| 12 | Next the code went to the FileCellRenderer class. | In this class we would say we found where the bulk of the change needs to be but now we need to go through the other renderers to find how to had a column. |
| 13 | After going through the VFSDirectoryEntryTableModel class we found where the column names are added. |  |

**Time spent (in minutes):** 120

# 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 | This will change the VFSDirectoryEntryTableModel because the path column has to be added after the name column. | *To track the classes that could be impacted by the change.* |
| 2 | The logic in the FileCellRenderer class will change to account for the path column too because we don't think the path column should be considered an extended attribute | *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 | The properties files for languages needs to include the path property. |  |
| 4 |  | *Because the class deals with the rendering the panel instead of the table of the schedule* |

**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.

|  |  |  |
| --- | --- | --- |
| Step # | Description | Rationale |
| 1 | make new width functions. in the FileCellRenderer class to split the name width and path width. Change done in FillCellRenderer.java | The original code was making the name column as wide as the path plus name and that measurement needed to be split between two columns now. |
| 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):** 30

# 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 first added the path property to the jedit\_en.props file | This is how the code knows which columns to add. |
| 2 | Added an index == 1 case for the path property lookup. Change in VFSDirectoryEntryTableModel.java | Since we consider path to be a core property, because it was taking away from the name property, it needed a special case for always being the 2nd element. |
| 3 | Changed the Favorite’s label to store just the name not the path. Change in FavoritesVFS.java |  |
| 4 | Then changed all the column index offsets for the getExtendedAttribute lookups | Index 0 was for name and we made index 1 for path |
| 5 | This is when we realized that these changes affected the general browserView. We had to account for the index offset by 2 in a few places throughout the VFSDirectoryEntryTableModel class. | We got index out of bounds exceptions |
| 6 | We then changed the FileCellRenderer class in the getTableCellRendererComponent because this is where the actual text was changed |  |
| 7 | This is where we inserted the new width methods because the path column was not wide enough and the name column was too wide |  |

**Time spent (in minutes):** 150

# 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 | Added favorites directories then clicked on the edit favorites button | The browser opened with the favorites directory showing. It also showed the new path column with the correct size. |
| 2 | Move the browser to the root directory. | This will test a directory with more directories and files. It also tests to see if the general browser view broke. |
| 3 |  |  |
| 4 | *...* |  |

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

# Timing

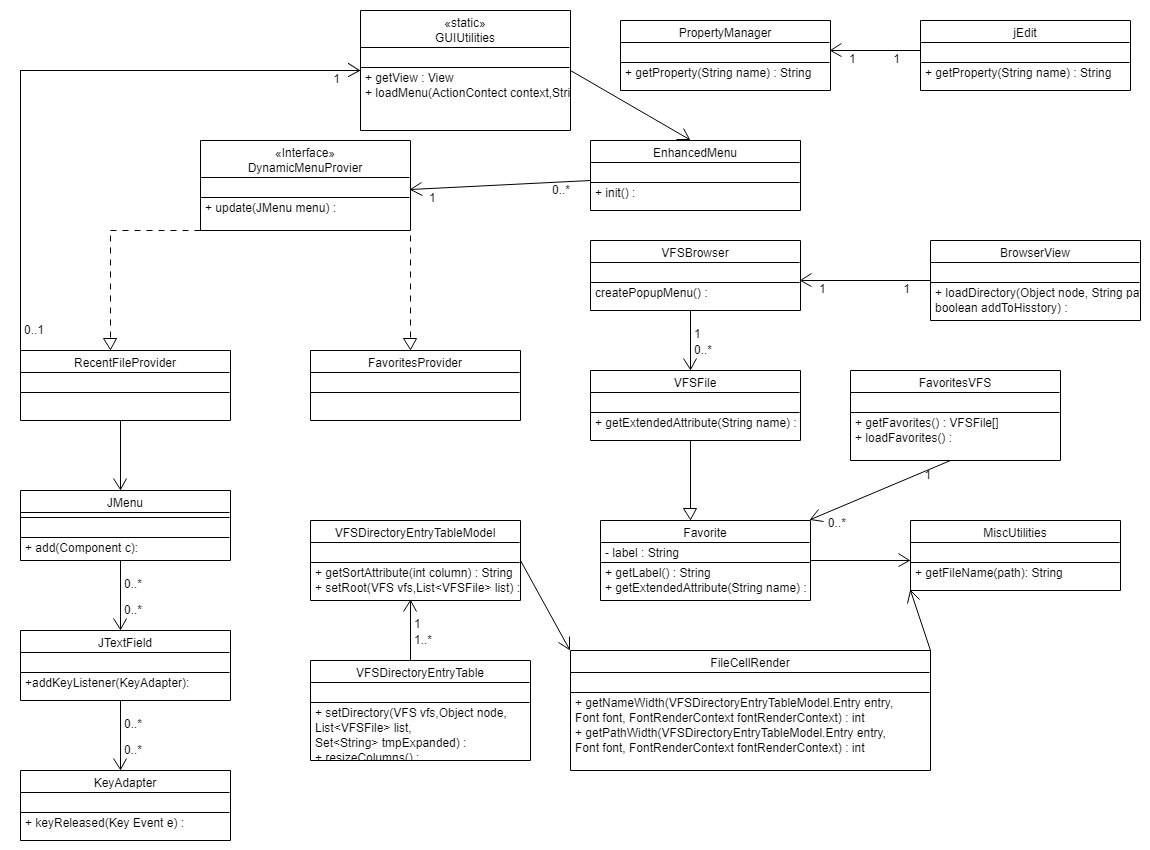
Summarize the time spent on each phase.

|  |  |
| --- | --- |
| Phase Name | Time (in minutes) |
| Concept location | 120 |
| Impact Analysis | 20 |
| Prefactoring | 30 |
| Actualization | 150 |
| Postfactoring | 0 |
| Verification | 20 |
| Total | 340 |

# 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 hard because it was hard to track the properties from the prop file. We also did not know much about providers and Models and how they relate to a view in the UI. After it was apparent that the VFSDirectoryEntryTableModel held the data and the FileCellRenderer add the data to the UI it was easier to see how the UI was populated. The other difficulty was coming up with an accurate Impact Analysis. Some impacts were unknown until changes were made.

Classes and methods changed:

* org/gjt/sp/jedit/browser/FileCellRenderer
  + public Component getTableCellRendererComponent(JTable table,Object value,Boolean isSelected, Boolean hasFocus,int row,int column)
  + added:
    - int getNameWidth(VFSDirectoryEntryTableModel.Entry entry, Font font, FontRenderContext fontRenderContext)
    - int getPathWidth(VFSDirectoryEntryTableModel.Entry entry, Font font, FontRenderContext fontRenderContext)
* org/git/sp/jedit/browser/VFSDirectoryEntryTable
  + private void resizeColumns()
  + private void saveWidths()
* org/git/sp/jedit/browser/VFSDirectoryEntryTableModel
  + public int getColumnCount()
  + public String getColumnName(int col)
  + public String getSortAttribute(int column)
  + public String getExtendedAttribute(int index)
* org/git/sp/jedit/io/FavoritesVFS
  + Favorite(String path,int type)