Change request log

# Team

Specify team name, and full names of the team members and their role.

# 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 | *We ran the system* |  |
| 2 | *We interacted with the system: after logging in we entered the schedule screen.* | *To get familiar with some of the features of the system and identify the screens or graphical elements we had to change.* |
| 3 | *We searched for "Block\*Time" using the regular expression feature of the IDE's search tool.* | *Because we identified a button in the screen called "Add Block Time"* |
| 4 | *From 15 results, we clicked on the class ScheduleUI. The class was inspected using the Structure window of the IDE.* |  |
| 5 | *We inspected the class ScheduleGUI. We went to this class using the dependency navigator of the IDE editor.* | *We noticed that the method ScheduleUI.createGUIcomponents was creating an instance of ScheduleGUI.* |
| 6 | *...* |  |
| 7 | *We inspected the class StudentBlock.* | *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 | *We added a break point in the line 34 of the method displaySchedule of the class StudentGraph. We ran the program in debugging mode.* | *We were not sure if this method had to be changed, therefore we decided to debug.* |
| 9 | *We marked the class StudentGraph as "located".* | *We confirmed this class had to be modified.* |

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

# 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 made a list of methods called by StudentGraph.displaySchedule* | *To track the classes that could be impacted by the change.* |
| 2 | *We inspected the class GUIRenderer. Such a 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):** x

# 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 | *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):** 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 created the class ScheduleHandler.* | *We realized that class Schedule was too big to add more code in it, that's why we created a new class.*  *The responsibility of the class is...* |
| 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):** x

# 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 | *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):** 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 | *Test case defined:*  *Inputs: …*  *Expected output:...* | *This is the regular expected behavior.*  *The test passed.* |
| 2 | *Test case defined:*  *Inputs: …*  *Expected output:...* | *This is an exceptional behavior (when values are larger than 100 characters)*  *The test passed.* |
| 3 |  |  |
| 4 | *...* |  |

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

# Timing

Summarize the time spent on each phase.

|  |  |
| --- | --- |
| Phase Name | Time (in minutes) |
| Concept location |  |
| Impact Analysis |  |
| Prefactoring |  |
| Actualization |  |
| Postfactoring |  |
| Verification |  |
| Total |  |

# 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 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.*