Change request log

# Team

reNo – Matt Dragan and Ben Sattelberg.

# Change Request

Change request 3.3

The Merge module throws an exception upon attempting to merge page ranges that intersect (see Figure 7). You are requested to fix this issue by allowing intersection of ranges during the merging operation.

# 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** | *Build and run PDFSam* | *Become familiar with the software.* |
| **2** | *Using the eclipse search tool search “merge”* | *The module that handles merging two pdf files together will likely have merge in the name.* |
| **3** | *Explore the MergeModule.java class* | *This module may contain or be related to the location that needs to be changed.* |
| **4** | *Explore other files in the merge directory* | *MergeModule.java wasn’t immediately helpful in determining where a change needed to be made, but the concept may be located in the other files.* |
| **5** | *We looked through the pdfsam-basic, pdfsam-core, and pdfsam-fx modules.* | *It was pretty hard where the actual pdf processing was occurring, so, like the previous change request, we spent a significant amount of time looking through other directories. Separately, we realized that much of the functionality is handled in the “sejda” package. We were still unable to easily find the functionality we needed so we didn’t include all of the substeps. PDFsam is build on top of the sejda functionality.* |
| **6** | *Checkout the* [*www.sejda.com*](http://www.sejda.com/)*website.* | *We wanted to get an idea of what sejda is and how it works.* |
| **7** | *Investigate some of sejda’s functionality through github* | *We wanted to see what sejda was doing, but this was challenging because we could only easily look at the modules that were imported into the pdfsam code. Navigation to different parts of sejda was difficult and, at this point, not very helpful in locating concepts.* |
| **8** | *Revisit the merge module* | *To further understand how the module works as well as understand how the system interacts with the sejda modules.* |
| **9** | *Look at the rotate module* | *As per the Professor’s suggestion, looking at the rotate module would likely be helpful. This helped us gain an understanding of what was happening. The apply function in the RotateSelectionPane.java seemed to be where the main part of the processing takes place.* |
| **10** | *Check out the MergeSelectionPane.java module* | *If this module has similar functionality to the RotateSelectionPane.java we might be able to identify where the change needs to be mode.* |
| **11** | *We notice there is a function called in what seems to be the processing step of the data called toPageRangeSet()* | *For this change request the goal is to allow for overlapping page ranges, which is related to how the program gets a range of pages.* |
| **12** | *Use the Eclipse search tool to search for the pageRangeSet() function which brought us to the ConversionUtils.java.* | *Understanding this function might point us in a direction where were will be able to implement the change request.*  *This proved to be helpful in understanding how the page ranges are split into tokens.* |
| **13** | *Checkout the toPageRange() function called by the pageRangeSet() function.* | *This is a sejda module. This function parses a page range and returns a PageRange type which includes a start point and an endpoint. We were able to determine, also, that when a single page is given this function gives a page range which just includes the single file because the ranges are inclusive.* |
| **14** | *Do another search using the Eclipse search tool for pageRangeSet()* | *The pageRangeSet() function that we found contains the functionality that we are looking for, but the method called in the MergeSelectionPane.java file doesn’t take any parameters. ‘*  *This lead us to the SelectionTableRowData.java module. This is where the apply() function in MergeSelectionPane.java calls a version of the toPageRangeSet() function which calls another version of the function which returns a set of page ranges.* |
| **15** | *Add print statements to the toPageRangeSet()* | *We used these statements to verify that this function was being called when a merge was executed.* |
| **16** | *We tentatively marked the MergeSelectionPane.java module as located.* | *This class seemed like the class that most likely needed to be modified.* |

**Time spent (in minutes):** ~420

# 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 apply in the MergeSelectionPane.java module* | *To track the classes that could be impacted by the change.* |
| **2** | *We inspected the MergeParametersBuilder.java. It was not marked for changes.* | *This object is being used by the apply function, but the functionality is embedded in the sejda functionality so making changes to MergeParametersBuilder.java would require us to make changes to sejda.* |
| **3** | *We inspected the class ConversionUtils.java. This class was marked to change* | *The functionality in the apply method will be difficult to change, but making changes to the ConversionUtils.java will simplify the process. Also, adding functionality here to allow for overlapping page ranges could be reused for other functionalities of PDFsam if desired.* |
| **4** | *We inspected the SelectionTableRowData.java class. This was marked for change.* | *Having a toPageRangeSet() function and a toPageRangeSetMergeOverlappingRanges() is a simple change and can be extended to other modules in PDFam. There may be methods of updating the code that wouldn’t require this addition, but for ease of use and potential reuse this seemed reasonable.* |
| **5** | *We made a list of classes that may call MergeSelectionPane.java’s implemenation of apply.* | *To track classes that could be impacted by the change. However, it appears that this functionality isn’t used in any of these classes. This method may be called by an underlying sejda module, but these changes shouldn’t affect the underlying module.* |

**Time spent (in minutes):** 37

# 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 added a method in ConversionUtils.java called toPageRangeSetMergeOverlappingRanges(String selection) which has similar functionality to the original toPageRangeSet(String selection). There is some unnecessary code reuse here, but implementing the the required functionality without code reuse is not very elegant.* | *We need to create a Set<PageRange>, but the creation of the Set must account for potential overlaps in page ranges. Adding this functionality easily allows us to handle overlapping pages in any PDFsam module that uses it.*  *This may not be the best way to handle this change, but I think it is fine because it supports easy reuse if we want to extend this functionality later.* |
| **2** | *Added a method in SelectionTableRowData.java called toPageRangeSetMergeOverlappingRanges() which calls the toPageRangeSetMergeOverlappingRanges(String selection).* | *We added this because of the way the setPageRange() is called in the merge module we need the SelectionTableRowData module to have an additional member function that can handle page ranges that may overlap.* |
| **3** | *After the previous changes we ran the system and the unit tests.* | *We wanted to make sure that the refactoring of the code doesn’t affect any of the functionality.* |
| **4** | *Committed our changes with git.* | *Just in case we need to revert our changes.* |

**Time spent (in minutes):** 35

# 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** | *In MergeSelectionPane.java’s apply method I changed the call to toPageRangeSet() to toPageRangeSetMergeOverlappingRanges()* | *The prefactoring allowed this small change to the MergeSelectionPane.java class to merge overlapping page ranges meanwhile maintaining all of the original functionality of merge without affecting other parts of PDFsam.* |
| **2** | *We ran existing test cases.* | *Make sure that the changes made to the apply function do not affect the original functionality. These changes did not affect the original functionality of merge.* |
| **3** | *Ran functional testing to ensure that expected behavior occurred.* | *Merging with overlapping page ranges occurred successfully.* |

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

# 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 did not find postfactoring necessary.* | *The changes made were relatively isolated. It might be a good idea to change the functionality so there isn’t as much rewritten code, but that solution was less readable and less elegant so we believe having the additional code is acceptable.* |

**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.

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

|  |  |  |
| --- | --- | --- |
| **Step #** | **Description** | **Rationale** |
| **1** | *Code Review between members of the group.* | *Make sure that the changes made were reasonable and matched the change request.* |
| **2** | *Functional testing: we input two files and tried different variations of overlap in the page ranges.*  *Test1: Test for intersecting ranges (ex. 1-4, 3-6)*  *Test2: Test for nested ranges (ex. 1-4, 3)* | *Tests if the intended behavior occurs based on the change request.* |
| **3** | *Functional testing: input two files and merge without overlapping page ranges.* | *Ensure that original functionality works as expected.* |
| **4** | *Add two unit tests which test the two cases presented in step 2.* | *Providing automated testing is useful for ensuring validity. For this particular area of the code the tests were written in a way that was easy to understand making adding these tests simple.* |

**Time spent (in minutes):** 32

# Timing

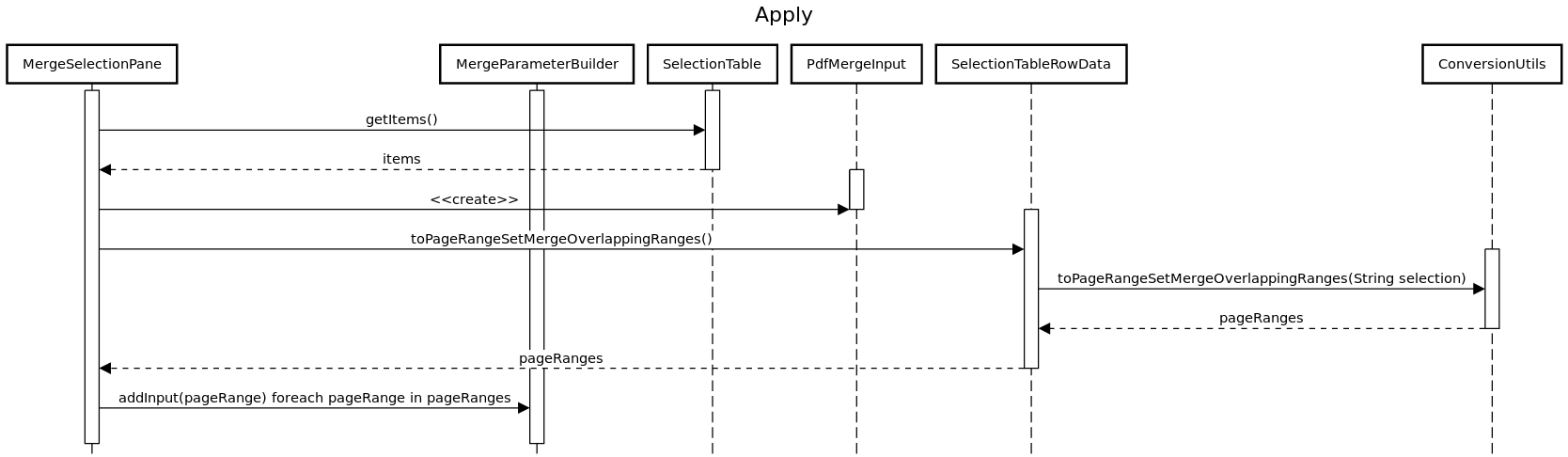
Summarize the time spent on each phase.

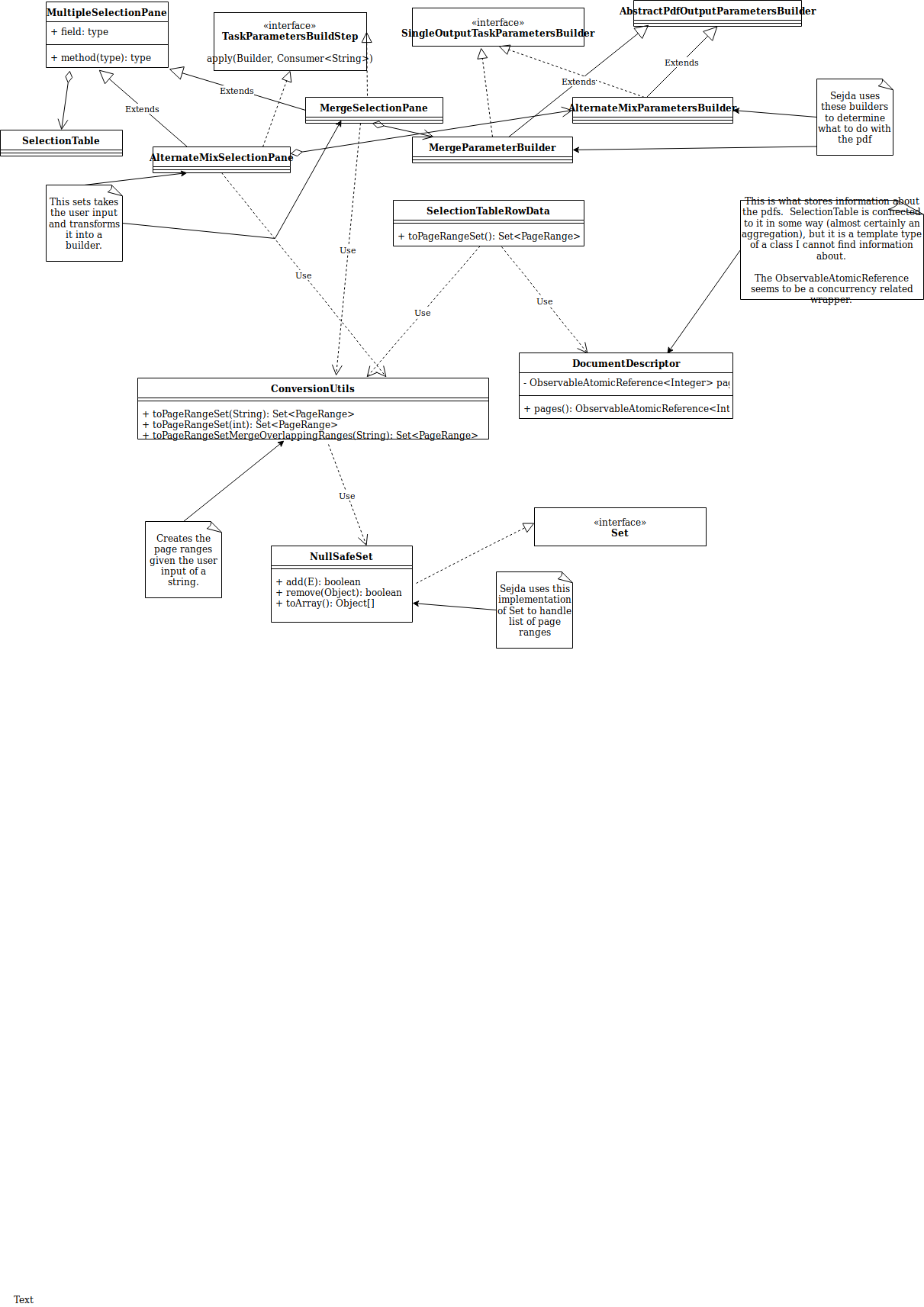
|  |  |
| --- | --- |
| **Phase Name** | **Time (in minutes)** |
| Concept location | 420 |
| Impact Analysis | 37 |
| Prefactoring | 35 |
| Actualization | 20 |
| Postfactoring | 5 |
| Verification | 32 |
| **Total** | 549 |

# 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.

*Concept location for this change was a significant challenge. Neither my partner nor I have much experience with large such large scale programs and usually not in an object-oriented language like java. It took us a while to understand what was going on. Additionally, understanding the divide between PDFsam and sejda was not easy because the inheritance of methods and use of sejda’s exterior library made tracking down functions somewhat difficult. We don’t believe our implementation of this change request was handled in the most elegant way. This change doesn’t do a great job of reusing code. However, we weren’t sure how to make this change in a more elegant way because of our lack of familiarity with the overall structure. We used Eclipse search functionality for both concept location and impact analysis. We tried to use Jripples for impact analysis, but neither of us could successfully get it installed which made impact analysis more challenging. Automated validation was relatively simple because it was easy to build on previous unit tests to test the new functionality. The UML class diagram was created using draw.io and the sequence diagram was created using sequencediagram.org.*

*One strange behavior we noticed was that some of the unit tests related to the desktop client application would fail on a clean clone from git. Wecommented out the tests that caused these errors.*

*Matt implemented the changes with input from Ben. Matt wrote the documentation and built on the UML diagram created by Ben, and created the sequence diagram.*

*Classes and methods changed:*

* *pdfsam-parent pdfsam-mergesrc/main/java/org/pdfsam/merge/MergeSelectionPane.java*
  + *void apply(MergeParametersBuilder builder, consumer<String> onError)*
* *pdfsam-core/src/main/java/org/pdfsam/support/params/ConversionUtils*
  + *Set<PageRange> toPageRangeSetMergeOverlappingRanges(String Selection)*
* *pdfsam-fx/src/main/java/org/pdfsam/ui/selection/multiple/SelectionTableRowData*
  + *Set<PageRange> toPageRangeSetMergeOverlappingRanges()*