Change request log

1 Team

Brent Staab

2 Change Request

Change Request: ps2
The Merge module throws an exception upon attempting to merge page ranges that intersect. The change is to allow intersections of ranges during the merge operation.

Example: For the range “1-10,5,20-25”, the resulting file should contain the pages 1 to 10, followed by page 5, followed by pages 20 to 25.

3 Concept Location

1st attempt – This is the approach I took for the first solution (union of pages)

Step #	Description	Rationale
1	*Code inspection* - *Could not find what I was looking for*	*Try to find location in code based on static analysis*
2	*Run program in debug mode* - *Put break points at various points likely to be executed*	*Try to identify code by seeing which functions were being called on a running system*
3	*The program architecture is such that each function has a corresponding folder. For this change request, I started looking in src/main/java/org/pdfsam/merge*	*Since I didn't know where to look, I started in the folder named Merge*
4	*The main file is MergeModule.java. In this file, I placed a few break points and found that the initModuleSettingsPanel() function to be the correct path*	*This was based on trial and error since I didn't have a better idea*
5	*I followed the code down to src/main/java/org/pdfsam/merge/MergeSelectionPane.java then SelectionTableRowData.toPageRangeSet and ultimately to ConversionUtils.toPageRangeSet()*	*Just followed the path until I found what I was looking for*
6	*In the Eclipse IDE, I was able to inspect the 'pageRangeSet' value and see that it contained the intersecting ranges*	*Verified the processed data (ranges) had the offending values*
7	*I experimented with other modules and found the Extract functionality handles intersections without issue. I spent a little time looking for that implementation but was not successful.*	*I was hoping to find an existing solution that could be leveraged for this change request.*
8	*Task finished*	*While there may be a better place to implement or possible existing solution to leverage, I believe I found the location where I will implement the change request functionality*

Time spent (in minutes): 120

2nd attempt – This is the approach I took for the updated solution (allow duplicate pages)

Step #	Description	Rationale
1	*Using the Eclipse IDE, I put a breakpoint in the function I created for my 1st attempt* - *org/pdfsam/support/params/ConversionUtils.java* ○ *postProcessPageRangeSet()* *then run an example*	*I knew my code was being executed and functioning as I expected, so I thought it would be a good place to start looking for a different solution.*
2	*The stack trace in the debug view listed the path taken to get to my function. There were three entries for the MergeModule and one for the MergeSelectionPane. Since the MergeSelectionPane was the closest/most recent relative to the postProcessPageRangeSet() call, I focused on that*	*Focus on the function calls with 'Merge' since this is the module we ultimately need to fix.*
3	*Via the debugger, I found that the MergeSelectionPane's apply method was being called. (org/pdfsam/merge/MergeSelectionPane.java)*	*Focus on the specific 'Merge' function being called*
4	*Based on code inspection, I found that all page ranges listed on a given line were being added with the corresponding file name. Created a hypothesis that I could add a "file, PageRange" pair for each range defined and this would mimic the behavior found during experimentation*	*During experimentation, I observed that you could define intersecting pages if they were defined on separate lines* *1) File.pdf, range '1-4'* *2) File.pdf, range '2'* *Would result in a merged pdf with pages 1,2,3,4,2*
5	*Task Finished*	*While there may be a better place to implement or possible existing solution to leverage, I believe I found the location where I will implement the change request functionality*

Time spent (in minutes): 60

4 Impact Analysis

Step #	Description	Rationale
1	*The MergeSelectionPane.apply() function is called once for each Merge request and thus needs to be changed*	*This function processes the range associated with each PDF listed in the Merge module GUI*
2	*The new logic is like the old logic. It is a drop-in replacement and will be called at the same and with the same dependencies.*	*I'm not changing the basic logic (call stack), just enhancing the behavior*
3	*The amount of time to execute the new logic should be negligible. There may be more entries for the downstream processing to handle, but it's not expected to cause problems.*	*This is an assumption based on my anticipated implementation.*

Time spent (in minutes): 10

5 Prefactoring (optional)

Step #	Description	Rationale
1	*N/A*	*N/A*

Time spent (in minutes): 0

6 Actualization

Step #	Description	Rationale
1	*In the Concept Location phase, I identified the MergeSelectionPane.apply() function as the place to start*	*This function processes the files and ranges described in the Merge module file list.*
2	*The existing logic converts the ranges into a list of PageRange objects. All ranges for a specific list item are stored with the corresponding file name.*	*Just an observation of existing functionality*
3	*Decided to create a list consisting of a file name and at most one PageRange object. If a file has more than one range defined, there will be multiple entries for the file but with unique PageRange objects.*	*This seems to mimic the observation from concept location where intersecting pages could be defined if they were on separate lines.*
4	*Create a function called postProcessPageRangeSet() to handle intersections*	*It's good programming practice to contain the changes in a function. This allows you to quickly identify the functionality and remove if necessary.*
	For each item in the table - *Strip leading and trailing spaces* - *Get the file name* - *Get the set of PageRanges*	*For items with non-whitespace characters, get the file and PageRange information*
	For each PageRange in the set of PageRanges - *Add PageRange to PageRangeSet* - *Create new PdfMergeInput with the file name and PageRange* - *Add to list of file+page range set*	*Create a new file+PageRange object for each PageRange in the set. This is to mimic the behavior described earlier where intersecting pages were allowed as long as they were on different lines.*

Time spent (in minutes): 120

7 Postfactoring (optional)

Step #	Description	Rationale
1	*N/A*	*N/A*

Time spent (in minutes): x

8 Validation

Step #	Description	Rationale
1	*Test Case: Range with intersecting page* ***Input**: 1-3,1* ***Expected Output**: Pages 1,2,3,1*	*Test valid input* *The test passed*
2	*Test Case: Range with intersecting pages* ***Input**: 1-3,1,3,2* ***Expected Output**: Pages 1,2,3,1,3,2*	*Test valid input* *The test passed*
3	*Test Case: Unbound range with intersecting page* ***Input**: 6-,10* ***Expected Output**: Pages 6,7,8,9,10,10*	*Test valid input* *The test passed*
4	*Test Case: Intersecting unbound ranges* ***Input**: 1-3,2-4* ***Expected Output**: Pages 1,2,3,2,3,4*	*Test valid input* *The test passed*
5	*Test Case: Intersecting pages* ***Input**: 8-,9-* ***Expected Output**: Pages 8,9,10,9,10*	*Test valid input* *The test passed*

Time spent (in minutes): 30

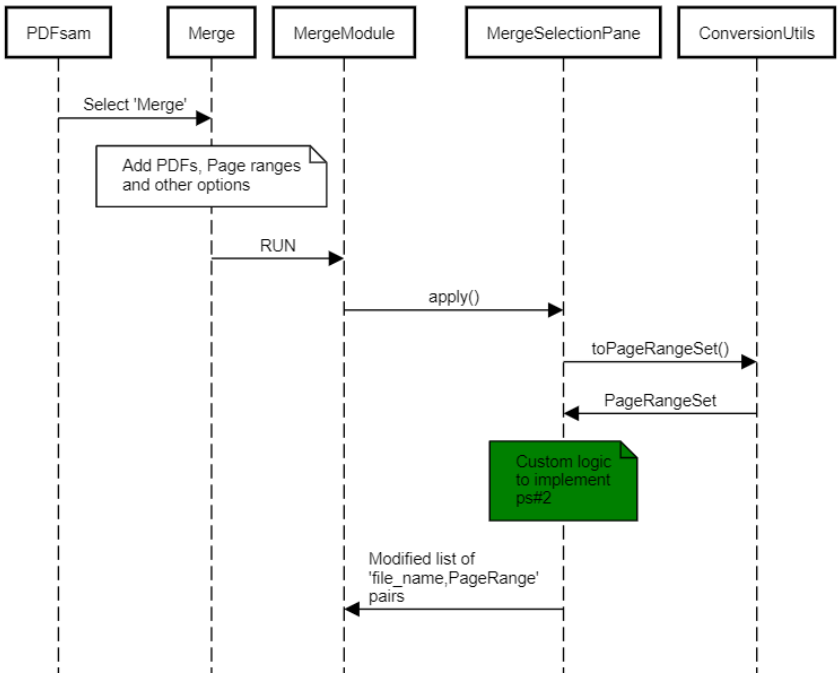
9 Timing

Summarize the time spent on each phase.

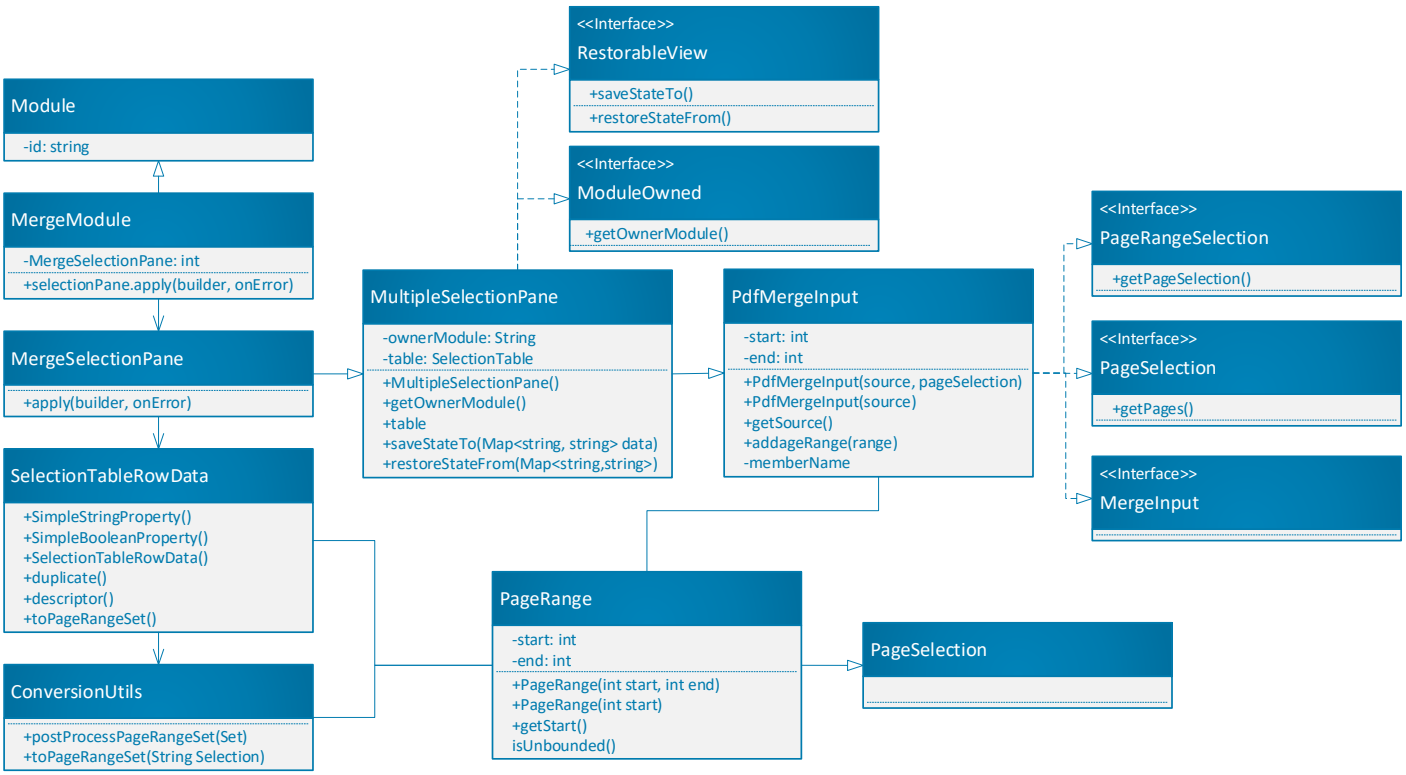
Phase Name	Time (in minutes)
Concept location	120 + 60
Impact Analysis	10
Prefactoring	0
Actualization	120
Postfactoring	0
Verification	30
Total	340

10 Reverse engineering

UML sequence diagram



Partial UML class diagram



11 Conclusions

For this change, the concept location was more difficult because I had to use a debugger to identify the location in code. However, once identified, the impact analysis was straight forward because I found that the change was isolated to a very specific chunk of code. The actualization of the change took about as long as I expected because it was largely refactoring existing functionality. Also, the Eclipse IDE made suggestions regarding things that needed to be imported so I didn't have to wait for compilation to identify them. The testing was performed using the Eclipse debugger. I would provide various ranges and execute a merge. If the output wasn't what I expected, I would step through the code, inspect variable values and logical path.

Classes and methods changed:

- *Classes*
  - *MergeSelectionPane*
    - *Pdfsam-merge/src/main/java/org/pdfsam/merge/MergeSelectionPane.java*
- *Methods*
  - *apply(MergeParametersBuilder builder, Consumer<String> onError)*