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:

- Range: 1-10,20-30,4
- Issue: Page 4 is implicitly defined in the range 1-10

Example:

- Range 50-60,55
- Issue: Page 55 is implicitly defined in the range 50-60

3 Concept Location

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

4 Impact Analysis

Step #	Description	Rationale
1	*The ConversionUtils.toPageRangeSet() function is called once for each pdf listed in the Merge list and thus needs to be changed*	*This function handles the page ranges listed in the Merge module GUI*
2	*The new logic will only be called when a merge operation is executed, but this is consistent with the current behavior.*	*I'm not changing the basic logic (call stack), just enhancing the behavior*
3	*The amount of time added due to new functionality will scale based on the number of documents to be merged and their specific ranges.*	*This is an assumption based on my anticipated change.*

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 ConversionUtils.toPageRangeSet() function as the place to start*	*This function processes the range(s) described in the Merge module*
2	*The existing logic processes each range for a single PDF file and converts to a PangeRange object. The collection of PageRange objects are stored in set and returned to the caller*	*Just an observation of existing functionality*
3	*Decided to create a post processing step after all ranges have been converted to a PageRange but before the set is returned to the caller*	*In order to identify an intersection, we need all ranges defined for a file*
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.*
5	*First step is to create a union of all pages in all the ranges. Duplicate entries will be discarded.*	*This seemed like the simplest and most straight forward approach*
6	*Sort the list so we'll have a monotonically increasing list of pages, or empty list if no rage was specified.*	*This will be needed for downstream processing*
7	*Find the lowest starting page number for all unbounded ranges.*	*An unbounded rang has a start and no stop which means all pages after the start will be included. Only the lowest start matters, all other unbound ranges are implicitly included.*
8	*Remove any pages that intersect with the unbound range from the sorted union.*	*Since the unbound range implicitly includes the intersecting pages, they are not needed*
9	*Build a new set of PageRanges based on the files left in the sorted union*	*The original list may not be valid any more due to the aggregation of page ranges*
10	*Need to handle various conditions* - *Unbound range* - *Not contiguous range of pages* - *Last page in the list* - *etc*	*Need to handle all combinations of page ranges*

Time spent (in minutes): 180

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

Time spent (in minutes): 45

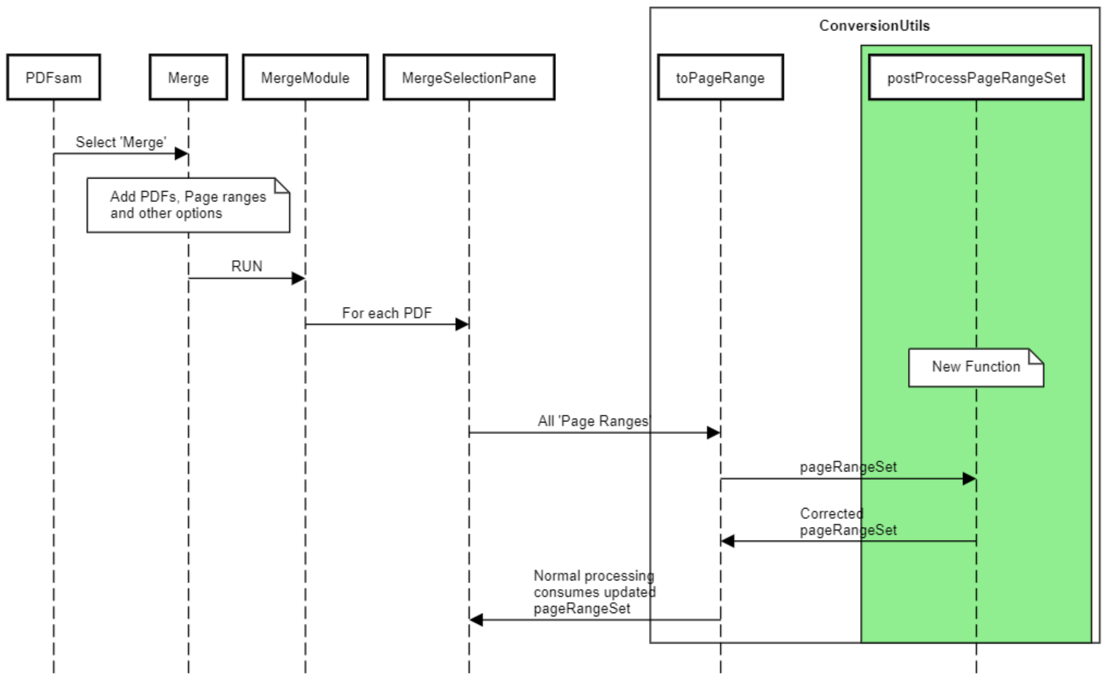
9 Timing

Summarize the time spent on each phase.

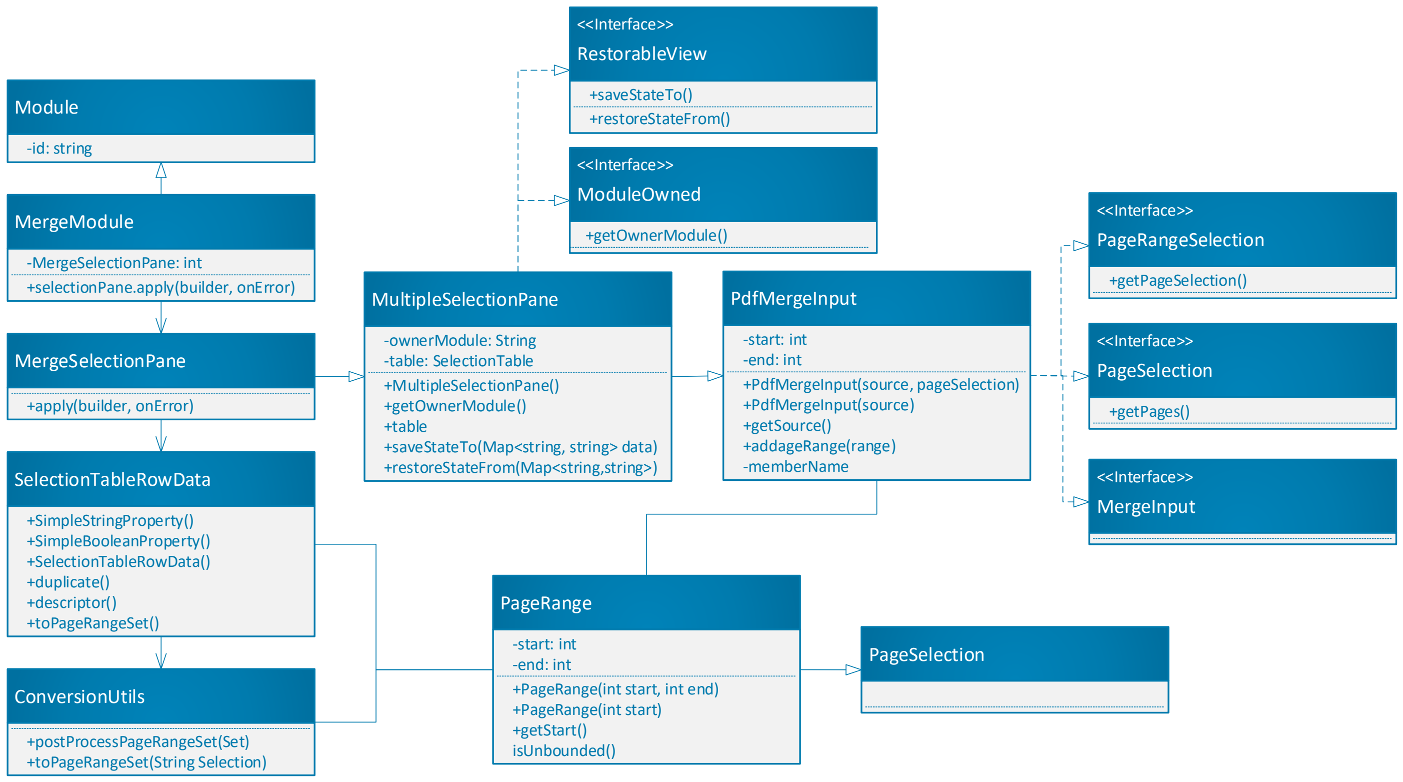
Phase Name	Time (in minutes)
Concept location	120
Impact Analysis	10
Prefactoring	0
Actualization	180
Postfactoring	0
Verification	45
Total	355

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 bit of functionality. The actualization of the change took longer than anticipated because there is a plethora of valid combinations to consider. 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*
  - *ConversionUtils*
    - *pdfsam/pdfsam-core/src/main/java/org/pdfsam/support/params/ConversionUtils.java*
- *Methods*
  - *Modified*
    - *toPageRangeSet(String selection)*
  - *New*
    - *postProcessPageRangeSet(Set<PageRange> ranges)*