Change request log

# Team

Team Badvision, consisting of Brendan Robert, the lead janitor.

# Change Request

PS2 – PDF Sam request to support overlapping page ranges

# Issue selection / Initiation

From assignment 1, I knew that PDF Sam had undergone an evolution in which the code was split into very separate modules. Most of the underlying model objects and PDF processing itself was moved into [Sedja.org](http://sedja.org/). Because of this I picked an issue that looked like something that either was the fault of the UI or was at least significantly influenced by the UI. I was hoping that the error message in #ps2 was overzealous validation code. That was unfortunately not the case, though the first hypothesis of UI-side repair was upheld.

## Getting started

* Checked out v4.0.5 release tag
* Raise issue in git project: <https://github.com/badvision/cs515-801-s20-robert-pdfsam/issues/1>
* Created feature branch for ps2 feature request (using git flow conventions)

# Concept Location

Concept location was relatively straight-forward but I did have to bounce around references until I finally located the utility class responsible for parsing user input.

|  |  |  |
| --- | --- | --- |
| Step # | Description | Rationale |
| 1 | *Reviewed stack trace from bug report* | *Stack traces can sometimes pinpoint the exact source of an error, and that saves a lot of time during concept location.* |
| 2 | *Stack trace was no help, decided to review Merge sub-module to see how table UI is handled* | *I already knew the application was well separated and anything related to this issue would be in either the Merge module or one of the common/core modules it depends on.* |
| 3 | *Found the view of the table extends a view in the JavaFX views module. I was able to pick out the class quickly because the classes were well-named.* | *I decided to look at what callback functions were registered on that column, as those would likely have to interpret user data and make decisions about it.* |
| 4 | *Eventually worked way through code to find the ConversionUtils class in org.pdfsam.support.params*  *using IDE to navigate via “jump to declaration” quickly.* | *Because of clean separation of concerns, the code was rather literate and easy to follow. I didn’t have to do a whole lot to understand it.* |

**Time spent (in minutes):** 30

# Impact Analysis

The impact analysis found that very few parts of the program would need to be affected, and that there was only one anticipated ripple effect: Page ranges entered anywhere in the application would follow this new behavior of converting into a normalized page set. This spill-over effect would need updates to documentation to prevent user confusion, but ultimately could be considered a positive side-effect.

|  |  |  |
| --- | --- | --- |
| Step # | Description | Rationale |
| 1 | *I looked to see how many other places in the code called ConversionUtils methods for parsing page order and found only one place related to the merge bug, and corresponding unit test code for that class itself.* | *I wanted to rule out, if possible, the need to perform any extra validation testing.* |
| 2 | *There were no other “to change” areas to explore after that, but I realize that this repair is only localized to PDF Sam, not to other software using the Sedja.org module that does the real work.* | *This concern means that for other software to share the improvement, this fix would ideally be refactored into Sedja.org instead.* |
| 3 | *Target solution identified: update parser for page ranges (as string) into a set of range objects; allow parser to optimizes the user input as if they typed in a pre-merged, properly ordered (aka normalized) set of pages.* | *This is a simple and elegant solution in that it only handles how user input is parsed, and also has few side-effects because it is a fairly isolated change.* |

**Time spent (in minutes):** 5

# Prefactoring (optional)

This code is super-clean. I didn’t have to move anything around before getting started on repair. There was already a unit test so my goal was to keep everything intact and add functionality only where I had to.

# Actualization

|  |  |  |
| --- | --- | --- |
| Step # | Description | Rationale |
| 1 | *Created test case in corresponding unit test for ConversionUtils to assert the desired behavior through combinations of page ranges, adjacent and overlapping.* | *Test-driven development is the best way to go for this kind of repair. We want to assert current tests continue to pass, and we want to assert our desired behavior of the parser and test in isolation because it is much faster than recompiling and executing the entire application.* |
| 2 | *Implemented code changes to ConversionUtils, tests passed but realized I was missing an edge case* | *I needed one more test case to make sure page ranges within unbound range would be ignored, for example “3-,4-5” should collapse to “3-“ because that is inclusive of all pages after 3.* |
| 3 | *Additional refactoring required a change to use AtomicInteger in new code, as evidenced by warning from IDE that variable should be declared effectively final.* | *AtomicInteger works as a final object which can still change value, thus satisfying the requirement needed to use and/or modify local variables within lambda expressions.* |
| 4 | *Tests pass, but realized another edge case was not handled, if there were multiple ranges, the second one was being truncated. Adjusted test to add "1-5,4-10,7,15"* | *The more tests the merrier. At this point I was also considering refactoring the test code as it was lacking elegance and overly-verbose.* |
| 5 | *Because I found this after completing the previous pull request, I raised the code changes in a new pull request* | *Following this process makes code changes easier to track over time.* |
| 6 | *All tests pass, performed manual testing and confirmed everything is working.* | *Always manually verify results! Unit tests can’t tell you everything.* |

**Time spent (in minutes):** 90

# Postfactoring (optional)

|  |  |  |
| --- | --- | --- |
| Step # | Description | Rationale |
| 1 | *The unit test code had a lot of copy/paste of long-winded boxcar-style method calls. I introduced a couple of assert methods to replace the box car code used throughout the unit test class.* | *Boxcar coding is okay in small doses, but if you copy/paste boxcar code multiple times it violates the “Do not repeat yourself” (DRY) principle of clean coding.* |
| 2 | *Confirmed that unit test code works.* | *Needed to ensure that code is working, even made it fail on purpose to check that the new methods work as expected.* |
| 3 | *Refactored the legacy unit test code to use the same methods* | *Consistency is key. This was optional but overall it is better that a unit test be as uniform as readable as possible.* |
| 4 | *Committed all code changes directly.* | *There was no impact to functionality, this was a preventative change.* |

**Time spent (in minutes):** 10

# Validation

Validation takes mere seconds thank to jUnit and the ability for Netbeans to execute individual tests, with the option to perform step-wise debugging. This was key in rapidly solving this bug.

|  |  |  |
| --- | --- | --- |
| Step # | Description | Rationale |
| 1 | *2-4,5-6* | *Adjacent ranges should be combined to 2-6, passed* |
| 2 | *2-4,5-* | *Adjacent bound and unbound ranges should combine as a single unbound range 2-, passed* |
| 3 | *2-10,5,7* | *Pages within existing bound range should be ignored, e.g. 2-10; passed* |
| 4 | *1-5,4-10,7,15* | *Overlapping ranges should merge, and non-adjacent ranges should be preserved, e.g. 1-10,15; passed* |
| 5 | *5,9-14,13-20,10,25* | *Overlapping range between other ranges should combine, and non adjacent ranges before and after should be preserved, e.g. 5,9-20,25; passed* |
| 6 | *1-4,3-6,7-10* | *Three overlapping ranges should combine to one, e.g. 1-10; passed* |
| 7 | *1-,3-6,7-10,100* | *Any ranges within an already established unbound range should be ignored; e.g. 1-; passed* |
| 8 | *10,9,8,7,6,5,4,3,2,1* | *Pages specified in any order, even backwards, should be understood as part of existing ranges and collapsed accordingly, e.g. 1-10; passed* |

**Time spent (in minutes):** 5

# Timing

Summarize the time spent on each phase.

|  |  |
| --- | --- |
| Phase Name | Time (in minutes) |
| Concept location | 30 |
| Impact Analysis | 5 |
| Prefactoring | 0 |
| Actualization | 90 |
| Postfactoring | 10 |
| Verification | 5 |
| Total | 140 |

# Reverse engineering

PDFsam service 
PDFsam merge module 
PDFsam corc 
PDFsam i18n 
POFsam split by bookmarks module 
PDFsam desktop client application 
POFsam simp e split module 
PDFsam split by size module 
POFsam alternate 
mix 
module 
PDFsam rotate module 
PDFsam extract module 
PDFsam JavaFx views 



**The Merge UI where the user interaction occurs is in the Merge module (1), in a view created by org.pdfsam.merge.MergeSelectionPane. This class extends from a parent in the JavaFX module (2) that defines the universal behavior of user selections, org.pdfsam.ui.selection.multiple.MultipleSelectionPane (which is used in other application features such as alternate mix, rotate, etc.) This uses a model object for the table data called SelectionTableRowData, which in turn calls a utility class in the Core module (3) called ConversionUtils.toPageRangeSet (which is called by a few other places, now all of which benefit from this feature extension.)­­**

# Conclusions

Overall, I feel that issue selection strategy (pick a UI bug) allowed me to identify a more solvable bug and compared to jEdit concept location only took a few minutes, not days.

There were many combinations of ranges, single pages, and unbounded ranges that needed test coverage in order to completely address this issue.

PDF Sam is well-designed code, and that greatly aids in concept location and impact analysis. The fact that core logic is in a separately maintained module adds essential complexity to the overall maintenance effort, and in this feature request, the code has become more fragmented because I fixed the bug in the one place indicated in the report, not in the core module that could benefit other software using that same core. The influence of Robert Martin (aka Uncle Bob) is unmistakable; this is very clean code.