Change request log

1 Team

Jared & Yashwanth

Jared Crouse: Change request log tracker

Yashwanth Virupaksha: Code navigation and changes

2 Change Request

#ps1

The Split by size module of PDFsam allows to split a PDF file in files of a given size (see Figure 5). Ideally, the created files should be smaller than the given size, but this does not always happen (see Figure 6). You are requested to fix this functionality, so that the created files never exceed the specified size, unless the created file contains a single page that by itself exceeds the limit size.

3 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 started PDFsam and tested splitting a document by size to see how it worked.*	*We wanted to test the existing functionality to understand how it worked before changing it.*
2	*The first place we started looking in the code was the pdfsam-split-by-size module where we found the following classes; SizeUnit.java, SizeUnitRadio.java, SplitBySizeModule.java, SplitBySizeParametersBuilder.java and SplitOptionsPane.java.*	*This seemed like an obvious place to start inspecting code to determine what will need to change in order to implement the functionality as requested in the change request.*
3	*The first file we started inspecting was SizeUnitRadio.java to see how the input document size was handled when the user toggles it.*	*We wanted to know how the user specified file size was handled.*

4	*The next class inspected was SizeUnit.java to see what happens there.*	*We decided to look into this class because SizeUnitRadio.java creates a SizeUnit object.*
5	*We noticed right away that the SizeUnit.java class has an enum also called SizeUnit which specifies conversion to bytes.*	*We knew something in this class would likely need to change to actualize the change request.*
6	*We noticed that both KILOBYTES and MEGABYTES are multiplied by a constant 1024 and knew that this would have to change*	*We were searching for variables that would need to change in this class.*
7	*We marked the class SizeUnit.java as "located".*	*We confirmed this class had to be modified.*

Time spent (in minutes): 64

4 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 examine the logic to convert Megabytes to kilobytes and kilobytes to bytes in SizeUnit.java*	*After examining the enumerated data structure we find that there's logic written to convert Megabytes to kilobytes and kilobytes to bytes. We are sure that we need to make changes over here to make the file size not exceed the given cutoff size.*

Time spent (in minutes): 11

5 Prefactoring (optional) SKIPPED

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): -

6 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 SizeUnit.java we changed the enumerated data type "SizeUnit" logic for converting Megabytes to Kilobytes as Megabytes = 1000 * Kilobytes and Kilobytes to bytes as Kilobytes = 1000 * bytes*	*We observed that when using the conversion logic Megabytes = 1024 * Kilobytes and Kilobytes = 1024 * bytes, the output documents size exceed the threshold. After changing the logic to the ones mentioned in the description we no longer notice the byte overflow above the cutoff.*

Time spent (in minutes): 5

7 Postfactoring (optional) SKIPPED

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): -

8 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	*We input a file of size 2.6 MB with split size as 500 KB. The first page is less than 500,and second, third, fourth and fifth above 500 KB, and the last two pages below 500 KB.* *Expected output: 6 files of size less than 500 KB*	*We see that PDFSam outputs 6 files with all files having file size less than 500 KB.* *Test passed.*
2	*We use a 6.1 MB pdf file containing text as input and split size as 1MB. All the pages in this document are less than 1 MB.* *Expected output: Three files of size less than 1 MB*	*PDFSam splits the pdf file into three files and all files are less than 1 MB.* *Test passed.*

Time spent (in minutes): 21

9 Timing

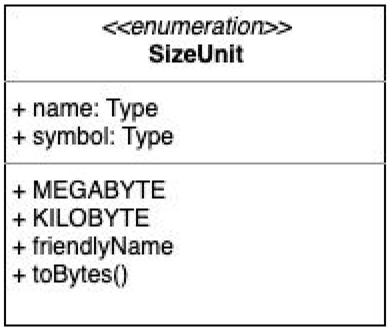
Summarize the time spent on each phase.

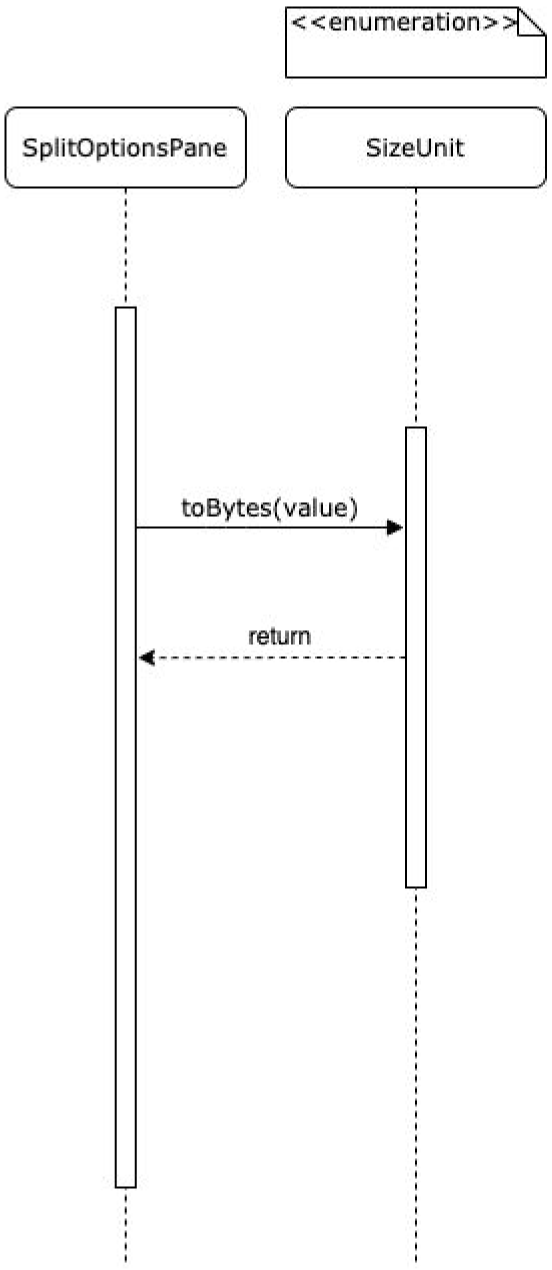
Phase Name	Time (in minutes)
Concept location	64
Impact Analysis	11
Prefactoring	-
Actualization	5
Postfactoring	-
Verification	21
Total	101

10 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.





11 Conclusions

This turned out to be one of the simpler changes to implement once we did the concept location. The actual concept location was the most challenging and time consuming part of the change request. Luckily it turned out that the change was fairly intuitive so we were able to narrow down code that may need to change to the split-by-size package which only contained five different classes. We initially started changing the values in SizeUnit.java to smaller numbers than 1000 and realized we were using the wrong numbers after some testing, Finally we used 1000 and were able to confirm that we had the right value after some brief testing.

Class methods changed:

- *pdfsam-split-by-size/src/main/java/org/pdfsam/splitbysize/SizeUnit.java*
    - *public enum SizeUnit - Changed*