Report for assignment 4

Project
Name: JabRef

URL issue 1: https://github.com/JabRef/jabref/issues/12565
URL issue 2: https://github.com/JabRef/jabref/issues/11998

URL Fork 1: https://github.com/elwi7331/jabref
URL Fork 2: https://github.com/mabeysekera/jabref

Open source software for managing bibliographic information (references).

Onboarding experience

We chose to abandon the last project because the issue tracker of the repository was unavailable to us. The repo for JabRef, on the contrary, uses the issue tracker extensively. The issues are marked with labels such as “good first issue”, which helped our issue selection. In addition, the maintainers have created a list of “Candidates for University Projects”, and that they encourage contributions from students attending courses similar to this one. In the end, we chose an issue marked “good first issue” but not in the latter list.

Similarly to the last project, we were able to build the software from source with relative ease. The main difference from the last project was that Gradle was used instead of maven (both are Java projects). Regarding tests, running ./gradlew test, 2 test cases fail: `pushEntries()` and `liveReloadCssDataUrl()`, at least on some of our computers. These tests fail on the main branch which we forked from, before we had done any changes to the code. After our changes, the two particular test cases still fail, though they seem to be irrelevant to our issue. On the testing performed in github actions however, the particular test cases do not seem to fail, indicating that the failures have to do with the execution environment.

Effort spent

	Elliot	Cintia	Adam	Oscar	Mary
1. plenary discussions/meetings;	5	5	5	5	5
2. discussions within parts of the group;	1	0.5	0.5	1	0.5
3. reading documentation;	2	2	1	3	2

4. configuration and setup;	1	1	1	1.5	1
5. analyzing code/output;	5	4	4	2	4
6. writing documentation;	2	1.5	2	2.5	5
7. writing code and running code	4	7	8	5.5	4
Sum (h)	20	21	21.5	19.5	21.5

Contributions

Elliot

- Proposed the issue that was ultimately chosen
- Forked repo and divided issue into sub issues based on initial assumptions about code structure
- Wrote the lines of code that are executed when “keep both” option is chosen by user
- Created diagram illustrating previous and updated behavior when encountering file name conflicts, from the view of a user
- Wrote about onboarding experience
- Review PR #7, #12 #11
- Fix errors that appeared from github actions test after initial pull request:
    - checkstyle import order
    - CHANGELOG.md formatting error
    - add strings to en_properties
- Review and approve PR with tests
- Fix errors that appeared from github actions after addition of tests:
    - Checkstyle import order and other issues (e.g. declaration order and excessive whitespace
    - `OpenRewrite`

Cintia

- Searched and proposed issues
- Implemented Tooltip for "Keep Both" Button
- Improved File Rename in message box
- Refactored Code for Maintainability
- Review and approve others’ PR
- Output patch to compare the change
- Write part of report for optional point 2 and 4

Adam

- Made textbox uneditable (was later made obsolete due to change of method)
- Wrote about feature requirements

- Wrote tests (PerformRenameWithConflictCheckTest)
- Wrote about the architecture and purpose of the system

Oscar
- Searched and proposed issues
- Changed the error message
- Created buttons
- Added changes to changelog
- Wrote team assessment
- Fixed problems introduced by merging
- Reviewed and created first proposed pull request

Mary
- Searched and proposed issues
- Found functionality “rename file to given name” and utilized it in appropriate place
- Wrote reflection SEMAT kernel
- Created UML diagrams
- Started to work on issue 2 since time was left
- Created issues (issue 2)
- Created tabs “Internal” and “BibTeX” (issue 2)
- Listed and displayed filenames and browse buttons. (issue 2)

Overview of issue(s) and work done

Title: Refine “File exists” warning

URL: https://github.com/JabRef/jabref/issues/12565

Files can be attached to bibliography entries. Refine the error dialog and implement additional measures for handling file naming conflicts.

Scope (functionality and code affected).

Requirements for the new feature or requirements affected by functionality being refactored

ID: R1
Title: Add "Keep both" functionality
Description: Implement a "Keep both" feature in the application, where JabRef automatically appends a suffix (e.g., (1), (2), etc.) to a file name when a conflict occurs. The function should use

```
org.jabref.logic.util.io.FileNameUniqueness#getNonOverWritingFileName
```

to generate unique names.

When the user hovers over the "Keep both" button, a tooltip should display the proposed new filename, with a placeholder {} replaced by a filename that doesn't yet exist (e.g., "Bogner (1).pdf" if "Bogner.pdf" already exists).

ID: R2
Title: Add "Provide alternative file name" button
Description: Add a new button labeled "Provide alternative file name" to the interface. This button should trigger a functionality that allows the user to rename a file to a custom name. The proposed name should appear in a text box pre-filled with the suggested filename, which can be edited by the user. This functionality should replicate the "Rename file to a given name" feature currently available in the context menu.

ID: R3
Title: Change dialog title from "File exists" to "Target file already exists"
Description: Modify the title of the dialog box that appears when a file conflict is detected. The new title should be "Target file already exists" to provide a clearer and more descriptive message for the user.

ID: R4
Title: Improve content display in the message box for long file names
Description: The content in the message box for existing files should handle long file names better. Currently, the content may look strange if the file name is too long. The message box should display the content as "Target file name {}", where {} is replaced by the actual file name, and the content should remain legible regardless of the length of the file name. Additionally, the content of this message box should be non-editable to prevent accidental modifications.

Project plan for testing

The primary goal of this testing project is to validate the behavior of the performRenameWithConflictCheck method in LinkedFileViewModel when handling file rename conflicts. The test cases ensure that the system correctly resolves conflicts based on user input and applies appropriate renaming strategies. This testing framework follows a unit testing approach using JUnit and Mockito to verify the correctness of file renaming functionality under different conflict scenarios. Mockito is used to simulate user interactions and system behavior. Dependencies are mocked to isolate and control test scenarios.

Tests
There were no tests prior to the ones we added. We added four tests to test for different scenarios of the function:

PerformRenameWithConflictCheckTest

all > test.java.org.jabref.gui.fieldeditors > PerformRenameWithConflictCheckTest

4	0	0	1.702s
tests	failures	ignored	duration

100% successful

Tests | Standard error

Test	Duration	Result
performRenameWithConflictCheckWhenKeepBothChosenKeepsBothFiles()	0.946s	passed
performRenameWithConflictCheckWhenNoConflictRenamesFile()	0.254s	passed
performRenameWithConflictCheckWhenOverrideChosenOverwritesFile()	0.249s	passed
performRenameWithConflictCheckWhenProvideAlternativeFileNameChosenUsesUserInput()	0.253s	passed

Optional (point 2): Relate updates to design patterns/refactor patterns

Context of Changes in Architecture:

- The changes primarily affect the GUI layer of JabRef, which is responsible for user interaction. The update ensures better user experience by refining the file conflict dialog, aligning with JabRef's existing modular MVC (Model-View-Controller) architecture.
- The update interacts with the Logic layer by calling FileNameUniqueness#getNonOverWritingFileName to ensure filenames remain unique without conflicts.

Relation to design patterns:

- Observer Pattern: The tooltip functionality is dynamically updated using JavaFX's event handling, reacting to user hover events.
- Strategy Pattern: The implementation allows different user choices (override, keep both, provide alternative name) to be handled via different execution paths, ensuring flexibility in behavior.

Optional (point 3): **Traced to requirements:**

- **test1** -> R1
- **test2** -> traced to a pre-existing requirement: no conflict of file names
- **test3** -> traced to a pre-existing requirement: if a file is being renamed to a file name that already exists, the user can overwrite the pre-existing file with the new one
- **test4** -> R2
- R3 and R4 are purely UI requirements and therefore don't have their own individual tests.

Optional (point 4): the patch is clean.

Changes to Code and Test Suite
Code changes

1. Improved File Rename Conflict Handling

    - Added options: "Keep both" and "Provide alternative file name" when renaming a file to an existing one.
    - Implemented tooltips on "Keep both" to show the new unique filename suggestion.
    - Refined the confirmation dialog UI to improve clarity and user experience.
2. Localization Updates
    - Added new localized strings

Test Suite Enhancements

1. New Test Class: PerformRenameWithConflictCheckTest.java

2. Test Scenarios:
    - No conflict: Successfully renames the file.
    - Conflict - User selects Override: Overwrites the existing file.
    - Conflict - User selects Keep both: Generates a unique filename.
    - Conflict - User selects Provide alternative file name: Renames the file based on user input.

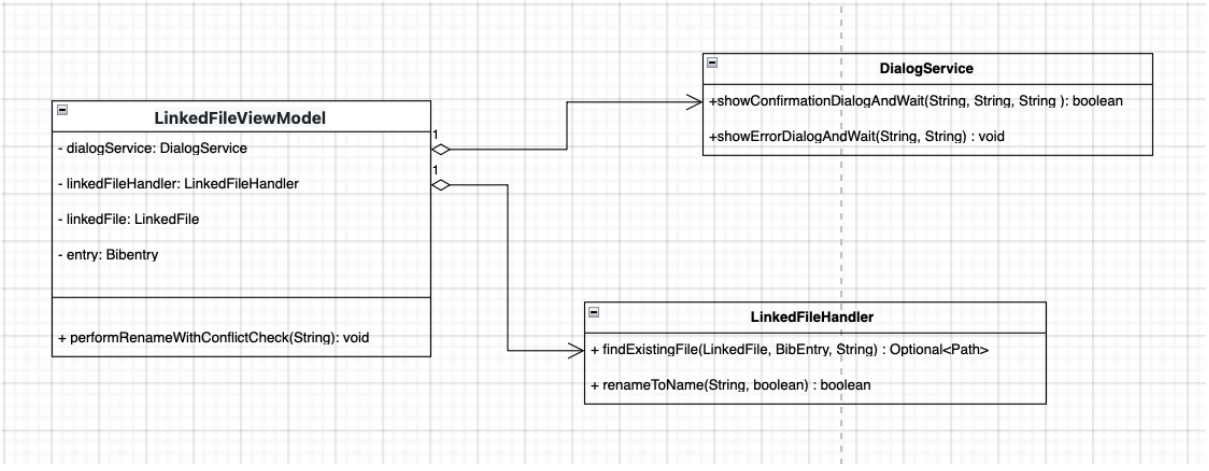
Patch
Command to show patch:
`git diff d8257d7 945a81f > changes.patch`

Our patch is clean as it removes obsolete code without commenting it out, avoids extraneous debug output, and does not introduce unnecessary whitespace changes. The modifications strictly focus on improving file rename conflict handling, ensuring clarity and maintainability without adding redundant elements.

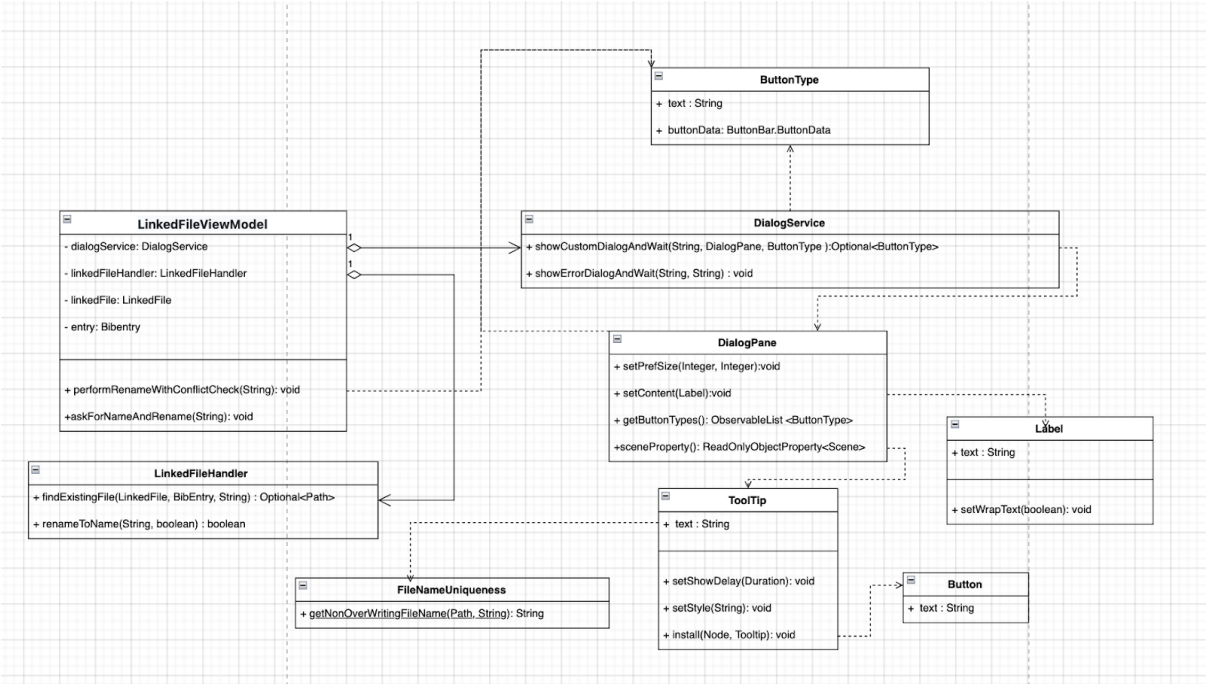
Optional (point 5): considered for acceptance (passes all automated checks).

UML class diagram and its description

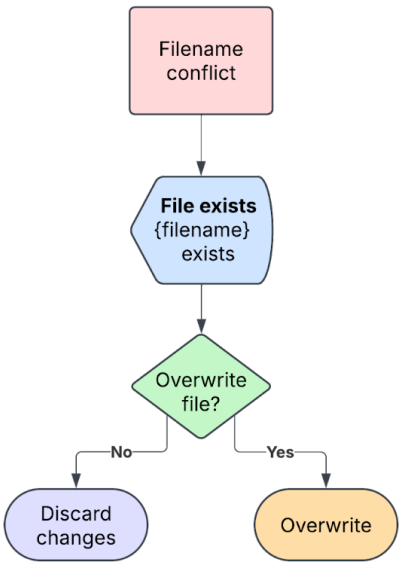
Before refactoring and adding features:



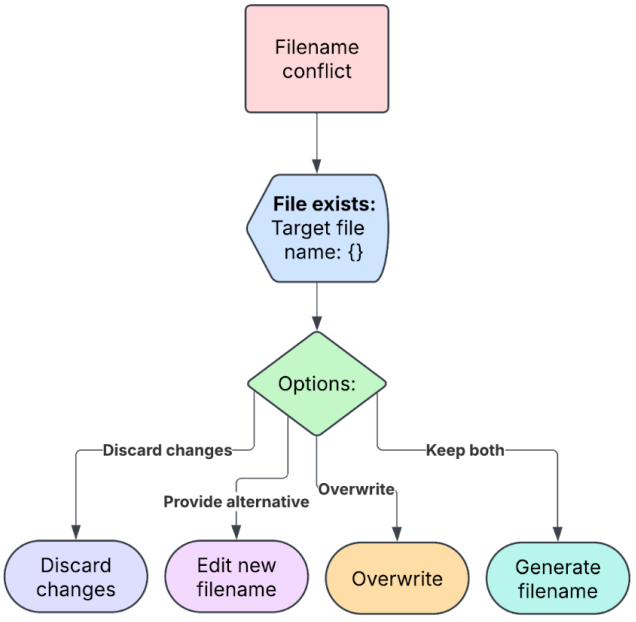
After refactoring and adding features:



Current behavior



Updated behavior



(Optional point 1): Architecture and system purpose overview:
JabRef is an open-source, cross-platform citation and reference management tool designed to help in organizing and citing bibliographic information. It uses BibTeX and BibLaTeX as its native formats, making it particularly suitable for LaTeX users.

Purpose and features:
JabRef allows users to manage their literature by offering the following functionalities:

- Collecting references: Users can add entries either manually or based on the reference text itself. They can also search across various online sources, import references in multiple formats, retrieve full-text articles, and extract metadata from PDFs.
- Organizing references: The tool allows users to manage the sources' metadata through a convenient UI. It also allows users to group their sources into hierarchical collections, organizing them based on keywords or tags. This in turn allows for advanced search and filter features. Users can also keep track of what they have read through filters such as relevancy, ranking, and priority.
- Citing sources: JabRef allows formatting references in numerous citation styles, including BibTeX and BibLaTeX support.
- Sharing and exporting: The tool contains built-in export options. It also saves libraries as text files, thus making them easy to be shared.
- AI functionality: JabRef has built in AI functionality, allowing users to use AI to generate a summary of any research paper and even discuss the papers with the AI assistant.

System architecture:
JabRef's architecture is structured into several layers to promote modularity and maintainability:

1. Model layer: This layer represents the core data structures, including bibliographic databases (BibDatabase), individual entries (BibEntry), standard field names (FieldName), etc… It contains quite minimal logic, focusing on data representation.
2. Logic layer: This layer serves as an intermediary. It handles operations such as reading, writing, importing, exporting, and manipulating the data defined in the model. It essentially acts as an API for the GUI layer to interact with the core functionalities.
3. GUI layer: This is the outermost layer. It manages user interactions and preferences. It depends on both the logic and model layers to present data and respond to user actions.

The GUI layer never directly modifies data but instead makes requests to the Logic layer and Model layer. Similarly, the Logic layer processes requests made by the GUI layer and interacts with the Model layer as needed. The Model layer finally stores and structures the bibliographic data but does not handle operations beyond data storage.

The advantages of this architecture include the modularity and maintainability of the system. The fact that each layer can be developed and maintained independently allows for easier debugging and refactoring. Another advantage is the ease of testing, allowing the Logic and

Model layers to be tested without requiring the UI. To add to this, the dependencies between these layers are directed towards the center, meaning that outer layers depend on inner layers but not vice versa. Additionally, the architecture includes JUnit tests to test for crucial dependencies between these layers, causing the build to automatically fail in the case of any violation.

For the issue we worked on, we dealt with the GUI layer - the layer that interacts with the user and their preferences, helping them to solve tasks.

Optional (point 6): How would you put your work in context with best software engineering practice?

SEMAT Kernel

Work assessment

1. **Initiated**
   The required result was clear through the assignment instructions as well as the provided instructions given in the issue. Here the priority lay in fulfilling the assignment instructions rather than the issue since different issues varied in complexity and effort. The priority was to at least partially implement the issue and document the work along the way. The initiator of the work was the creator of the issue. Risks like unclarity in instructions were consulted with the author of the issue. Funding was not applicable in this scenario.
2. **Prepared**
   We committed to solving the whole issue as described by the issue creator. Mandatory checks were agreed as acceptance criteria, like adding documentation to the repo's markdown file, manual testing of features added and inclusion of screenshots at pull requests. We ensured that team members had availability to resources used in the project by testing to build the work ourselves and setting up the development environment according to the manual. The work was broken down into parts between the team members, using the issue instructions as a guideline.
3. **Started**
   The different parts of the task were divided into issues inherent from the issue description. Team members accepted issues and distributed the work among themselves. Work was monitored through assigned code reviews for each pull request.
4. **Under Control**
   Issues were closed as tasks were completed. Unplanned work was distributed and defined during recurring meetings. Risks such as test failures were identified as soon as possible and mitigated. Measures of the team's progress can be found in the fork from the amount of closed and open issues.
5. **Concluded**
   The individual tasks of the issue have been completed and a patch has been submitted to the stakeholders. The stakeholders are yet to approve the resulting software system.

6. **Closed**
   Lessons have been recorded and analysed in the report and there are no significant uncompleted tasks.

Reflection and overall experience
Some of the benefits of this work was that the issue was predefined into subparts by the issue author which made it easy to understand the task and divide the work among the team members. Furthermore it was relatively easy to build the development environment and it didn't require any tricky dependencies. However this did vary depending on whether you followed the more detailed guide or not and which code editor you used. Each build of the project did however take a long time and since testing involved the GUI, it took some time to check whether the changes made in the code had the desired functionality. Another benefit was that the project we picked had good developer guidelines and clear instructions for acceptance criteria. The repo was also very active and authors could be contacted if guidance was needed.

A drawback was that the issue was a bit small in terms of workload for the group members. Therefore we distributed documentation and other tasks more fairly to compensate for hours spent. Additionally another issue was picked so that team members who had remaining time could work tasks from that issue. Another limitation was the Lack of Integration Testing. We only tested our changes in isolation.

As most statements were fulfilled for each criteria in the alpha "Work" from the SEMAT kernel, our group is at the stage "Concluded".

Requirements assessment

1. **Conceived**

The issue needs to be resolved since the previous error dialog wasn't as flexible and good. Improvements were required such as adding more selections and branches from that window. The users of the system are those who use Jabref and experience errors when renaming files into a defined pattern.

2. **Bounded**

The purpose and the extent of the issue was given through the issue description. The criteria for success was clear through the points given in the issue.

3. **Coherent**

The big picture of the requirements were clear and intuitive as a user of the system. An example of usage scenario was provided in the issue description. Conflicts didn't occur and unclarities were communicated and addressed with the author of the issue.

4. **Acceptable**

A solution has been submitted to the stakeholders but not yet approved.

5. **Addressed**

All the requirements described in the issue as well as mandatory checks were fulfilled. Stakeholders have not yet approved the patch to be operational.

6. **Fulfilled**

Can't be evaluated since the stakeholders haven't approved the patch.

Reflection and overall experience
Some benefits in regards to this alpha was that the requirements were clearly stated through the issue description. The expected behaviour could be understood well. Furthermore, a use case of the behaviour was also given. The main drawback was that we finished our work quite close to the deadline and therefore we only had time to submit our patch which is yet to be approved. We could have potentially fulfilled all the criterias of the alpha if we didn't have this time constraint.

Team assessment (Essence)

Seeded
Continuing off the basis of the two previous projects we proceeded with the task defined by the course. Our role distribution is loose and no one is solely responsible for anything. This choice was motivated by the small size of the team and the previously good collaboration.

Formed
The distribution of work was a bit harder on this project since we don't exactly know what we are getting into. We chose our first issue considering this and looked for an issue with clearly defined separate parts. Progress was steadily made and we were never waiting on any individual parts.

Collaboration
Setting up regular meetings was paced our work and communication was not an issue. Because of the small size of the group, unorganised communication over our Whatsapp and Discord groups worked perfectly fine.

Performing
The structure of the task made the work less linear than previously but the team's workflow adapted to it. A very small overlap in work occurred when first understanding the issue since some sub-issues were very small but otherwise no conflicts or back tracking occurred.

Adjourned
There are no comments to be made about this state of the team.