

**CST2550**

**Software Engineering Management and Development**

**Coursework 1 – Library Website**

**Submission Report**

**Autumn/Winter term**

**2023/2024**

**Date of Submission: 15/12/2023**

**Student ID Number: M00885478**

**Lab Tutor: Aditya Santokhee**

Contents

[Introduction 3](#_Toc156137887)

[Evaluation of Implementation 3](#_Toc156137888)

[Software Design 4](#_Toc156137889)

[System Testing 4](#_Toc156137890)

[Implementation 5](#_Toc156137891)

[Difficulties 5](#_Toc156137892)

[Conclusion 6](#_Toc156137893)

[Summary 6](#_Toc156137894)

[Limitations and Reflection 6](#_Toc156137895)

[Approach of future tasks 6](#_Toc156137896)

# Introduction

This report serves to provide an insight into the project of building a library system in C++.

The project involved designing (with given information) as well as building and implementing a library system capable of managing members and books. A header file was implemented for each class, and then the logic was done in a separate file, once again for each class.

Finally, all files are compiled with a main file to produce a system which runs indefinitely and allows a librarian to add members, issue books and facilitate the returning of books (while also calculating late fines) and read any csv. file provided, granted the csv file follows the correct format, without needing to change the code.

The project also made use of source control, to facilitate the writing of code and a make file was instantiated to compile the code into a working application, clean the compiled code if necessary and compile a test application which is used to ensure that the code is working as intended.

## Evaluation of Implementation

The library system was successfully implemented, and the code complies to produce a system that uses a menu approach, where the user selects a number based on the action they need to be carried out. The system also has comprehensive handling of incorrect inputs, with appropriate messages to point out the user’s error.

All the required functionalities are met, and the system assumes that it will never be shut off. The system also looks for a .csv file in its own directory, allowing the insertion of a new .csv file that follows the same structure without needing to update the code. A single make file is included.

Since the instantiation of a Librarian or any of the Librarian getters and setters are not part of the criteria, the decision was made to start the program with a hardcoded librarian called manager, who is used to carry out all the criteria in the coursework.

# Software Design

The provided UML was not fully adhered to.  
The following changes were made:

* Added a members list to the librarian class to store the members which are created. Without this (or any other method of storing the members), the system is believed to be impossible to implement as given by the UML.
* Changed the parameter type within the “setBooksBorrowed()” method (in the member class) from a Book object to a reference to a vector containing a list of books to better reflect the method name’s functionality. This change was brought about to ensure an easier way of borrowing and returning books for the member.
* Added a BookID parameter to the calcFine() method within the Librarian class in order to calculate the fine for a book that was returned. If only a memberID is used, calculating a fine is very difficult, and the addition of this bookID parameter streamlined the calculation process.
* A date abstract data type was instantiated to adhere to the UML referencing a Date class.

The system was designed around the given UML, changing as little as possible.  
The activity and use case diagrams designed for the project were adhered to, and the system carried out the actions specified in them.

# System Testing

The make file included in the logic folder also contains a test target, allowing the user to compile the test code should they wish to test the system.

The test code covers all the getters and setters, as well as all core library system functionality (including error handling). Additionally, it makes use of the header files in the logic folder to mitigate any unnecessary code duplication. Copies of the original logic files were made to add some methods to make the testing process easier, however no core functionality was changed.

Due to the nature of the methods, the test code instructs the user on what data to enter, and actively provides feedback on the given data.

The test code was written using the Catch2 framework.

# Implementation

Git was used to manage and monitor the coding aspect of the system. Regular commits were made to ensure that changes are not too drastic and commits which contained ‘broken’ code were carefully described for fixing in future commits. Branches were also used for potential fixes (where there was uncertainty regarding their feasibility), however most commits were made on the main branch as there was only one person working on the project.

A single make file was used to facilitate the compiling of code into object files and then into an application. The make file also allows the user to clean the directory of any object and application files, as well as compile the comprehensive test system.

# Difficulties

Throughout the project, the following difficulties were encountered:

* Keeping track of the members was found to be a major issue at the beginning of the project. This was resolved by adding a members vector to the librarian class.
* The borrowBook() method in the Book class did not synergise with the way the program was coded, and as a result, the decision to not include it in the implementation of the program was made.  
  It is still coded per specifications, just not used. The reason is that upon borrowing a book, the system looks it up in the .csv file, and instantiates a book object using the data, allowing the same book to be borrowed by multiple people, at the same time.  
  Instead, the system now loops through all members (very inefficient, but with the given variables and classes, this seemed to be the only option) and makes sure no one has borrowed the book before issuing.
* Some methods’ parameters seemed to provide too little information to carry out their required functionality. This was resolved by adding more parameters or changing the parameters.  
  This is seen in the Members class (setBooksBorrowed()) and the Librarian class (calcFine()).  
  SetBooksBorrowed() originally made it very difficult to integrate with borrowing and returning books, and the decision was made to change the parameter type to a reference to a vector containing all the books to set, aligning the function of the method more with the name.  
  calcFine() only taking a member ID made it very difficult to calculate a fine, as all the book data is linked to the book object, and retrieving that object from the member’s borrowed books would’ve been inefficient compared to just passing in the ID of the book being returned and locating it that way.

# Conclusion

## Summary

The system provided appears to be intuitive to use and meets all the given criteria. A single make file is included which can compile the system, system test code as well as remove all the compiled code and applications as needed. The test code covers a wide array of scenarios without compromising system functionality.

Difficulties were encountered, however most were easily circumvented and solved by slightly deviating from the provided UML without compromising system functionality.

Source control was used to facilitate the process, and regular commits were made to the project due to its size not allowing for a single sprint completion.

## Limitations and Reflection

Should this system have to be built again, more care should be given to storing the data to easily make use of some methods.

The system has a limitation when reading book names. Due to some books having a comma in their name, the system provides incorrect information regarding said books to the user. This is because of the way the system reads the csv file, and unfortunately due to time constraints this could not be fixed. However, it still takes note of the book ID and keeps track of when it is returned.

Additionally, deviating from the original plan should not be heavily scrutinized unless excessive and unnecessary, as problems which only appear when very deep into the coding process may appear and the only way to solve them (with the structure that is built) would be to slightly deviate from the original plan.

## Approach of future tasks

Time management was a large problem.  
The commits made were not evenly spaced, with most being made near the end of the project leading to unnecessary time pressure. This needs to be rectified in future projects as this tie pressure resulted in a system flaw.

Reluctance to deviate from the UML made the process longer than necessary, and simply deviating would’ve made the process smoother and more enjoyable.