Residency Day 2: Cross-Language Application Development

Book Cataloging System

Avijit Saha, Rinku Gopali, Sailesh Limbu, Shafan Nazeer Ahmed

University of the Cumberlands

Advanced Programming Languages (MSCS-632-M50)

Dr. Vanessa Cooper

04/04/2025

**Core Functionality Implementation:**

**GitHub:** <https://github.com/rgopali25573/Group-6-Book-Cataloging-System-/tree/main>

Today, our group successfully implemented the primary functionalities for the Book Cataloging System in both C# and Ruby. These included adding books, removing book entries, searching through the catalog based on key attributes (title, author, genre), updating and generating simple filtered reports. The goal was to ensure each core feature was fully functional GUI and responsive within the user interface, and by the end of the day, both versions of the application met that expectation.

**Use of Language-Specific Features:**

In developing the application, we consciously leveraged features that are unique or idiomatic to each language.

* **In Ruby**, we utilized **blocks**, **iterators**, and **dynamic typing** to handle collections in a more flexible and concise manner.
* **In C#**, we used **LINQ** for querying the in-memory data collections, offering a clean and efficient way to filter and retrieve records based on user input.

**Language-Specific Integration**

Our goal was not just to replicate features across two languages, but to allow each implementation to reflect the design philosophies of its language.

* The **C#** version took advantage of **object-oriented programming with strong static typing**, allowing us to build well-defined classes for handling book data and enforce structure.
* In **Ruby**, we leaned into the language's expressive syntax and **built-in data structures** like arrays and hashes, which made list manipulation straightforward and clean.

Here are some the screenshots we gathered from **Ruby** implementation:

**A screenshot of a computer program

Description automatically generatedFigure 1:** Book Class

**A screenshot of a computer program

Description automatically generatedFigure 2**: Class BookManager

**A screenshot of a computer

Description automatically generatedFigure 3:** JSON File

**Figure 4: hash for JSON implementation A screenshot of a computer program

Description automatically generated**

A screen shot of a computer

Description automatically generated**A computer screen shot of a program code

Description automatically generatedFigure 5: Loading/Saving Data from JSON**

**Figure 6: CLI Menu Implementation**

**A computer screen shot of a black screen

Description automatically generated**A black screen with white text

Description automatically generated**Figure 7: CLI Book Adding Output**

A screen shot of a computer program

Description automatically generated**Figure 8:** CLI Output

**Language-Specific Integration**

Our goal was not just to replicate features across two languages, but to allow each implementation to reflect the design philosophies of its language.

**Figure 9: GUI using gtk3**

**A screenshot of a computer

Description automatically generatedA computer screen shot of a program

Description automatically generatedFigure 10**: Adding and Removing Implementation

**A screenshot of a computer

Description automatically generatedFigure 11: Adding Book using GUI**

**Figure 12: Book Added**

**A screenshot of a book catalog

Description automatically generatedA screenshot of a book catalog

Description automatically generated** **Figure 13: Remove/Update from Selection**

**Figure 14: GUI Output**

**A screenshot of a computer

Description automatically generatedFigure 15: GUI Error Handling**

**Now, we will move towards the C# implementation:**

**A screenshot of a computer

Description automatically generated**

**Figure 16: Project Structure**

**A screen shot of a computer program

Description automatically generated**

**Figure 17: Book Model**

**A screen shot of a computer program

Description automatically generated**

**Figure 18: Add Book**

**A computer screen with many colorful text

Description automatically generated**

**Figure 19: Remove Book**

**A computer screen shot of a program

Description automatically generated**

**Figure 20: Search Book**

**A screen shot of a computer program

Description automatically generated**

**Figure 21: Validate add Book**

**A screenshot of a book catalog

Description automatically generated**

**Figure 22: Application**

**A screenshot of a book catalog

Description automatically generatedFigure 23: Book Added**

**A screenshot of a book catalog

Description automatically generated**

**Figure 24: Book removed**

**A screenshot of a computer

Description automatically generated**

**Figure 25: Delete book confirmation**

**A screenshot of a book catalog

Description automatically generated**

**Figure 26: Search book by Author**

**A screenshot of a book catalog

Description automatically generated**

**Figure 27: Search book by Name**

**Code Organization and Readability**

Both implementations were written with a strong focus on clarity and maintainability.

* In C#, code was organized using well-named classes and methods, consistent formatting, and in-line comments for major logic segments.
* In Ruby, we kept the code minimalistic but expressive, relying on Ruby conventions to maintain readability while avoiding unnecessary complexity and add inline comments for major logic segment.

**Testing and Bug Fixing**

We performed manual and automated tests to verify the accuracy and stability of core functionalities.

* For **C#**, we created basic test scenarios within the development environment, validating behaviors such as empty field handling, invalid input rejection, and case-insensitive searches.
* For **Ruby**, we used lightweight test methods to simulate user inputs and checked whether the output aligned with expectations.  
  We also used built-in debuggers and print statements to trace issues and fix any runtime errors identified during testing.