Residency Day 1: Cross-Language Application Development

Book Cataloging System

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

University of the Cumberlands

Advanced Programming Languages (MSCS-632-M50)

Dr. Vanessa Cooper

04/04/2025

**Initial Planning and Design**

**GitHub Repo:** [**https://github.com/rgopali25573/Group-6-Book-Cataloging-System-**](https://github.com/rgopali25573/Group-6-Book-Cataloging-System-)

**Application Design:**

On the first day, our group is focusing on defining the structure and key features of the Book Cataloging System. The application is to be developed in both C# and Ruby, allowing us to compare how each language approaches the same set of functionalities.

We began by outlining the core features of the system. Some of the common features to both C# and Ruby implementations are listed below:

* A simple GUI interface to interact with the book catalog
* Add new book entries with details such as title, author, genre, and publication year.
* Remove an existing book
* Search books based on title, author, or genre
* Generate simple report of books grouped by author or genre
* Persistent storage (in-memory or file-based for simplicity)

We considered the differences between 2 languages while designing the overall structure in each language. They are:

**C# specific design:**

* Use of Classes to represent books
* Lists or Dictionaries for book collections
* LINQ for querying the catalog
* Windows Forms or WPF
* Strong typing and exception handling
* GUI Events tied to methods (button click -> add/search/remove)

**Ruby Specific Design:**

* Use of Classes and Hashes/Arrays for book data
* Leverage blocks, iterators, and Enumerable methods
* Emphasize Ruby’s dynamic typing and expressive syntax
* Planning to use Shoes (simple GUI framework) or Tkinter or a lightweight CLI/terminal-based UI.

**Component Breakdown:**

We tried to analyze the system into manageable components, ensuring parallel development could occur across both language versions. These components included:

* **Data Model:** Structure for representing a book record. It will book entity with title, author, genre, year. Here C# will have Class with properties whereas Ruby will have Class with attr\_accessors.
* **Book Storage Management:** We will be using in-memory data handling for all book instances using arrays/lists.
* **Search and Filter Logic:** Algorithms for retrieving books based on queries and filters. We can use LINQ queries in C# and select, find, etc. in Ruby.
* **GUI/ User Interface:** A basic interface for interacting with the application. We will be using Avalonia UI in C# and Tk or CLI in Ruby.
* **Reporting Feature:** Grouping and displaying books by genre or author. We will use LINQ group-by in C# and group\_by method in Ruby.

**Task Assignment:**

We allocated tasks among team members based on individual strengths, experience, and learning goals:

|  |  |  |
| --- | --- | --- |
| Team Member | Role | Responsibilities |
| Sailesh | Developer | C# Implementation |
| Avijit | Developer | Ruby Implementation |
| Shafan | QA/Tester | Testing and debugging |
| Rinku | Project Manager & Integrator | Code Review and Coordinate tasks, maintain GitHub, document design & challenges |

**Timeline Creation**

To keep development on track, we established a project timeline with specific deliverables for each day:

* **Friday**: Finalize application design, define components, task assignment and set up GitHub repository.
* **Saturday**: Build and test core features in both languages.
* **Sunday**: Finalize remaining features, write documentation, and prepare presentation.

This timeline helped distribute the workload evenly and ensured all project milestones were met in a timely manner.

**Documentation of Design & Anticipated Challenges**

This project involves two parallel implementations of a book cataloging system—one in C# with a GUI (Avalonia UI) and one in Ruby (CLI or Shoes GUI). Both versions must support CRUD operations on books, search and filtering, and simple reporting.

A shared document was used to record all design decisions, brainstormed ideas, and notes on anticipated challenges. We were mostly concerned how we will be writing in code in each language since no one has worked on C# and Ruby before. Some of the Language-Specific Challenges are specified below:

* Complexities involved in developing GUI Based Applications
* Strong vs. dynamic typing
* Querying data: Use of LINQ in C# vs. Enumerable methods in Ruby
* Memory handling differences
* Error handling as C# has more structure with try-catch and begin-rescue blocks in Ruby

This documentation will serve as the foundation for our final comparison report and help maintain consistency across the two implementations.

**References**:

Martin, R. C. (2008). Clean code: A handbook of agile software craftsmanship. Prentice Hall.

Microsoft. (n.d.). C# programming guide. Microsoft Learn. <https://learn.microsoft.com/en-us/dotnet/csharp/>

Ruby Style Guide. (n.d.). The Ruby Style Guide. https://rubystyle.guide