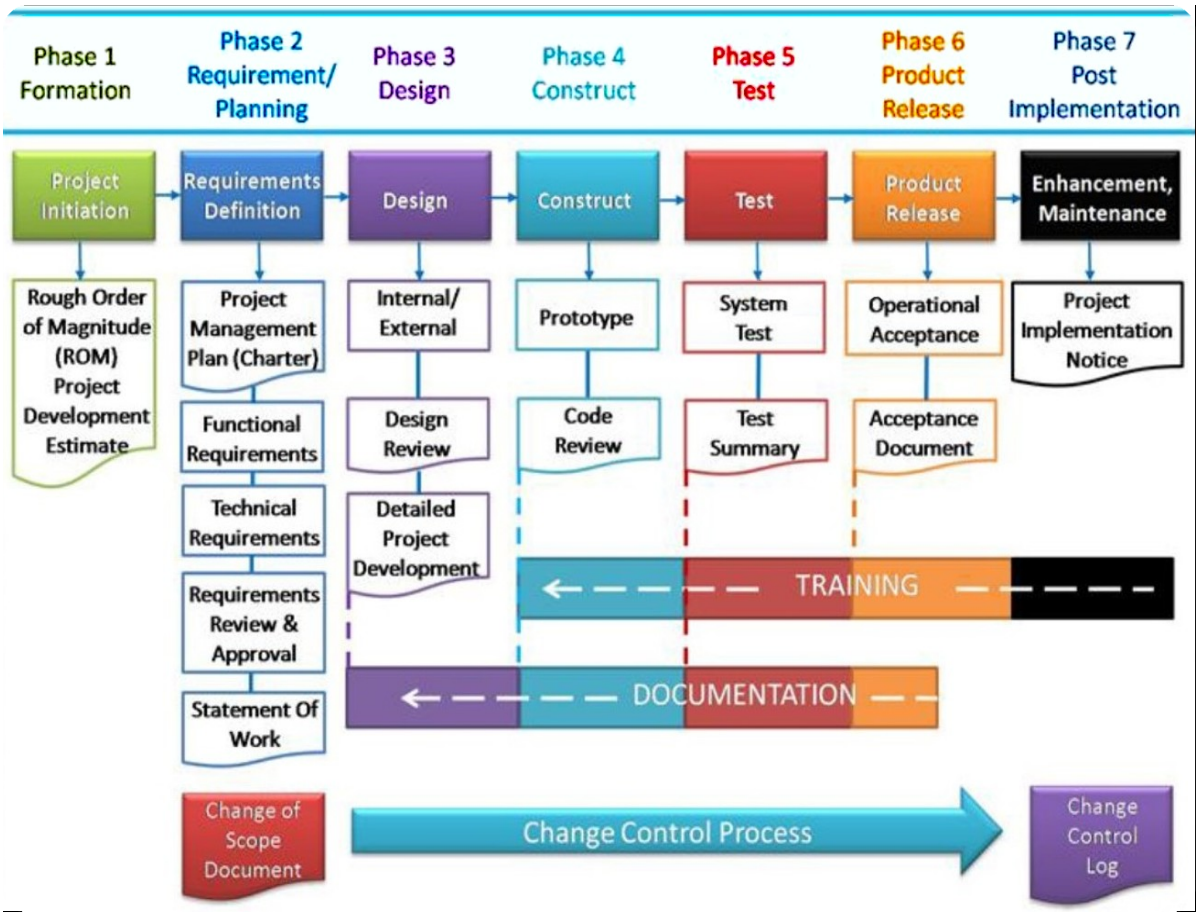
Tony Rojas

CEN 3024C – 13950
Aug. 31, 2025
Module 2: SDLC Part 1

Software Development Plan

Library Management System (LMS)

Software Development Life Cycle (SDLC)



This is the Software Development Life Cycle (SDLC) we'll be following for this project. Think of it as our overall game plan, our roadmap from start to finish. It's a proven method used (since 1970 @ Lockheed) in software development to keep things organized. As the image shows, the process has distinct unidirectional phases (cascading waterfall model), and we'll be tackling each of them in our plan, from figuring out what's needed, to building the app, testing it, and finally giving it to the librarian.

---

1. Introduction

The purpose of this software development plan is to break down how we're going to design and build a small Library Management System (LMS). The main goal of the LMS is to help a librarian manage a list of patrons. It's not meant to be a big, fancy web app or anything. It's just a simple console-based Java program that can load patron data from a file, let the user add or remove patrons manually, and show the current list. The customer scenario mentioned that this is for someone who just wants something basic and easy to use.

This document will outline the features we're building, how we gathered those requirements, the way we plan to implement the software step by step, and how we'll test it to make sure it works. We'll be focusing on keeping things practical and not overcomplicating it.

2.1. Requirements Definition

Here's what the LMS will need to do, based on the customer's scenario.

- **Functional Requirements**
  - **Store patron data**, including:
    - A unique 7-digit ID
    - Name
    - Address
    - Overdue fine (a number between $0.00 and $250.00)
  - **Add patrons from a text file**
    - One patron per line
    - Format: `ID-Name-Address-FineAmount`
    - No dollar sign in the fine field
  - **Add patrons manually** via the command line
  - **Remove patrons** by typing their ID
  - **Display all patrons** in a clean format
  - **Display a user menu** with options for the above actions
- **Non-Functional Requirements and Constraints**
  - The application will be **console-based**
  - It will be written in **Java**
  - All patron data will be stored **in-memory** (not saved to a database)
  - Each ID must be **unique**, and the system will check for duplicates

---

2.2. Requirements Gathering

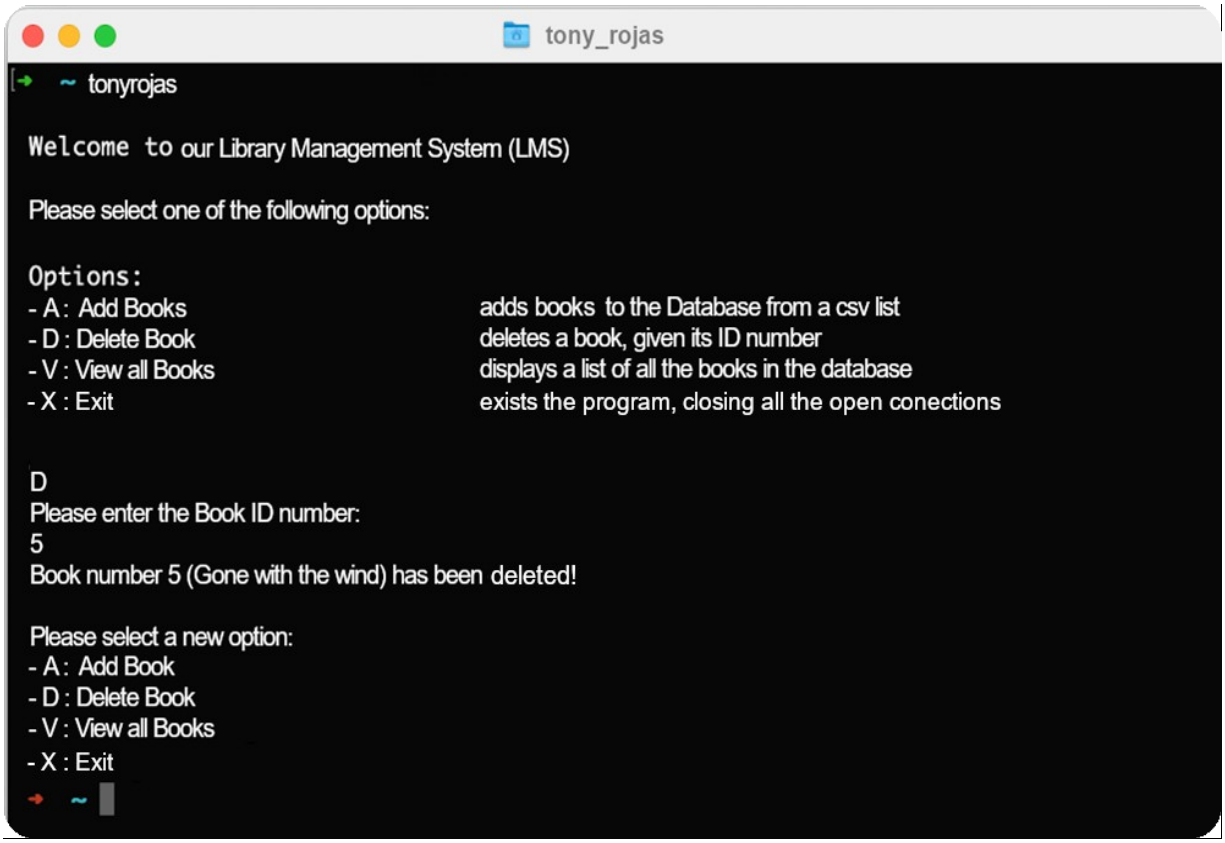
We're basing this project on a provided customer scenario that described what the librarian needed. From what we understood, the librarian doesn't want anything too technical—just something that works from the command line, doesn't require installing servers or databases, and lets them easily view, add, or remove patrons. They sometimes receive a full list of patrons in a file, so loading from a file is important. But they also want to be able to quickly add someone manually. And since the data doesn't need to persist between runs, using in-memory storage is okay for now, however saving it to a file seems more practical.

So our goal is to keep the interface friendly and straightforward, without extra stuff they didn't ask for. We're going to assume that we elicited (in a hypothetical interviewed and over-the-shoulder observations) more details from the librarian, and got a better idea of more nuanced series of expectations (e.g., reliability, clarity of UI, how errors should be handled, etc.).

---

3. Design: (an approximation of the User Interface)

As per this assignment's requirements the users must interact with the system using an on-screen menu. To be more specific, the users will interact with a command line interface (CLI) kind of like this one:



4.1. Implementation Plan

To build the LMS, we'll divide the work into two main classes: one to hold the patron info **`(Patron.java)`**, and another one to manage everything **`(LibraryManager.java)`**.

1. Patron Class

- File: **`Patron.java`**
- This class will include:
    - Fields for **`id, name, address, and fineAmount`**
    - A constructor to initialize a patron
    - Getters and setters
    - A **`toString()`** method for clean output

2. LibraryManager Class

- File: **`LibraryManager.java`**
- This will be the main app
- It will:
    - Keep an **`ArrayList<Patron>`** to hold patrons
    - Use **`Scanner`** to get user input
    - Display a looped **menu** with numbered options
    - Handle the following features:

Load from File

- Method: **`loadPatronsFromFile(String filename)`**
- Reads each line, splits it by `-`, and parses the data
- If the line is malformed or the fine is invalid, it skips the line and shows an error

Add Manually

- Method: **`addPatronManually()`**
- Prompts for ID, name, address, and fine
- Validates the inputs (7-digit ID, fine in valid range)
- Rejects duplicate IDs

Remove Patron

- Method: **`removePatron(String id)`**
- Loops through the list and removes the matching ID, if found

Display Patrons

- Method: **`displayAllPatrons()`**
- Loops through the list and prints each patron's info

Menu System

- Method: **`showMenu()`**

- Prints available options and prompts user input

4.2. UML Diagram:

This diagram shows the relationship between these 2 key objects: the **`Patron`** and the **`LibraryManager`** objects. They are composed of private attributes and public methods.

---

5. Testing Plan

We're keeping the testing simple for now. Since this is a small console project, we'll use **manual testing**—just running the program and checking that it behaves correctly in different situations. Here's a table of our basic test cases:

Test Case #	Feature	Input	Expected Output
TC-01	Display Menu	N/A	The main menu with all options should appear

Test Case #	Feature	Input	Expected Output
TC-02	Add from File	Valid file with 3 patrons	All 3 patrons are added, with a success message
TC-03	Add from File	File with malformed line (e.g., bad fine or missing dash)	Error shown for that line, rest of file still loads
TC-04	Add Patron Manually	Valid data (e.g., ID 1234567, Fine 15.50)	Patron is added, confirmation message shown
TC-05	Add Patron Manually	Invalid ID (e.g., 123)	Error message shown, patron is not added
TC-06	Add Patron Manually	Invalid Fine (e.g., 300.00)	Error message shown, patron is not added
TC-07	Remove Patron	Existing ID (e.g., 1245789)	Patron is removed, success message shown
TC-08	Remove Patron	Non-existent ID (e.g., 9999999)	Error message shown, list unchanged
TC-09	Display All Patrons	N/A	All current patrons are printed in clean format
TC-10	Add Duplicate ID	Same ID as existing patron	Duplicate message shown, patron is not added

We might add more test cases later, but these should cover the core features for now.

---

6. Deployment Plan

Since this is a simple console app, deployment is pretty straightforward. We don't have to worry about databases or web servers. The basic idea is just to get the program onto the librarian's computer so they can run it.

Step-by-step Deployment:

1. **Build the Project:** Once we're happy with the code and it passes all our tests, we'll compile the entire Java project into a single, runnable "`.jar`" file. We'll name it something simple like `LMS.jar`.
2. **Provide Prerequisites:** The librarian's computer will need to have a Java Runtime Environment (JRE) installed. We'll make sure to note this as a requirement. Most modern computers have it already, but it's an important detail.
3. **Deliver the File:** We'll just give them the `LMS.jar` file. They can save it to their desktop or a folder of their choosing.

4. **Provide Instructions:** We'll provide a simple text file with clear instructions on how to run it. The command they'll need to use will be something like:

- `java -jar LMS.jar`

  We'll also tell them how to put the "`patrons.txt`" file in the same directory so the program can find it.

That's it. No complicated setup, no installers, just one file and a quick command. It's a simple process that matches the simple nature of the application.

Final Thoughts

This is a simple project with a clear purpose: help a librarian keep track of people who use the library. We've broken it down into small, manageable parts and tried to keep the code easy to read and test. We're not using any databases or frameworks—just Java and a bit of common sense. If time allows, we might add an export/save feature in the future, but for now, we're sticking to what was requested.