Formal Documentation in Software Development Lifecycle
Software Engineering Lab (CS3273)
Assignment 1

Uttam Mahata (2022CSB104), Abhilash Kumar(2022CSB105)
Aryan Yadav (2022CSB106), Harsh Raj Sahu (2022CSB107)

January 26, 2025

Contents

1 Introduction

SDLC is a framework that defines the life cycle of a project from its initial stages to completion. "Life cycle" indicates the continuously evolving development, maintenance, and changes driven by user/client feedback. For example, when you first develop a website, you may have thought of a particular design and implementation. However, after the site goes live, based on user feedback or a comparison between your website and a competitor's website, you may think of a few changes to make your website unique and more user-friendly. This would mean another iteration or cycle of SDLC. While all the phases are not always essential, these are common phases for every software development project.

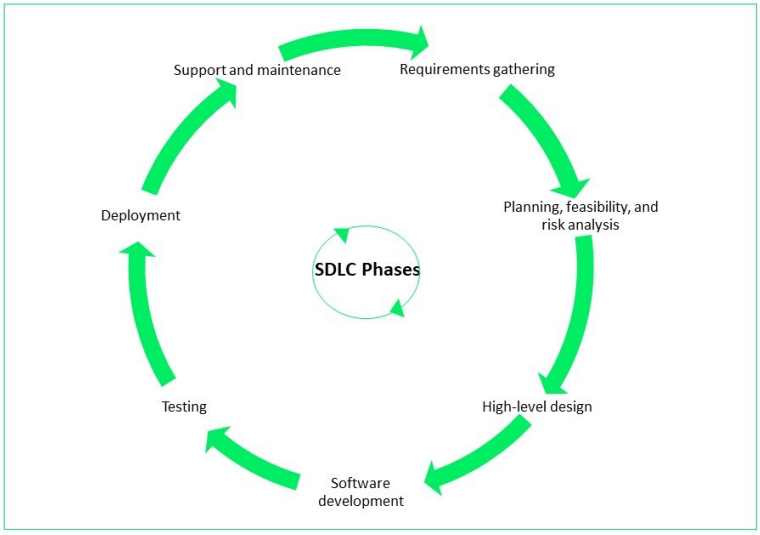

Figure 1: SDLC Life Cycle

2 Phases of SDLC

2.1 Feasibility Study

Description: In software engineering, a feasibility study entails planning and evaluating every facet of a project before moving forward. The procedure entails gathering information and assessing various project components in light of predetermined financial and schedule limitations standards. After evaluating all project-related factors, participants will come to a consensus on whether or not it is possible to finish this project within the allocated time frame and budget constraints. Participants will plan how to finish the project after deciding whether it is feasible. This will involve creating documentation for each process step using modeling techniques for software system design, dynamic flow mapping for computer application flow design, and other techniques.

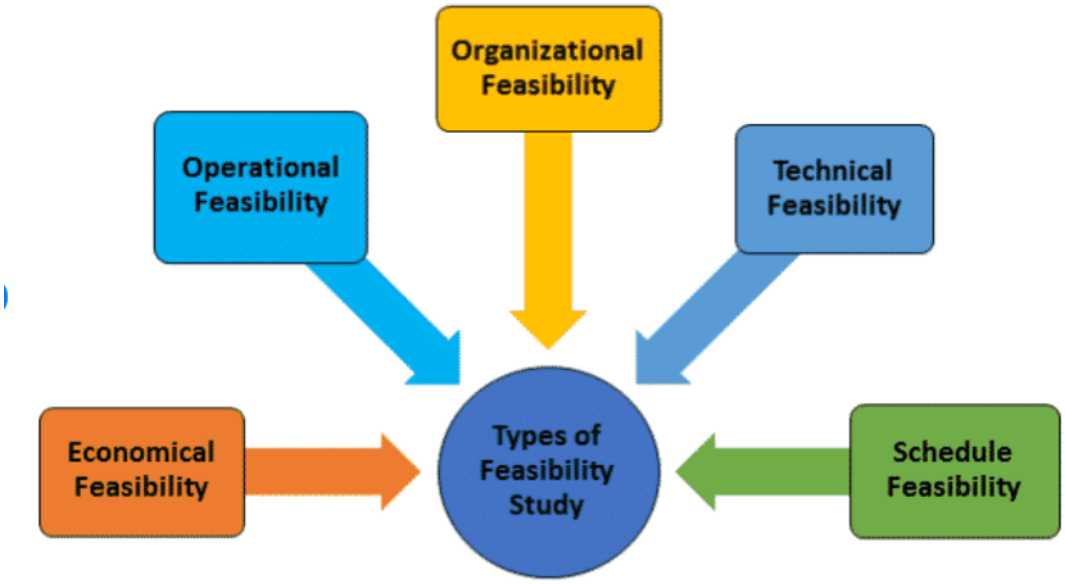

Figure 2: Feasibility Study

2.2 Requirement Analysis

Description: Requirement analysis is significant and essential activity after elicitation. We analyze, refine, and scrutinize the gathered requirements to make consistent and unambiguous requirements. This activity reviews all requirements and may provide a graphical view of the entire system. After the completion of the analysis, it is expected that the understandability of the project may improve significantly. Here, we may also use the interaction with the customer to clarify points of confusion and to understand which requirements are more important than others.

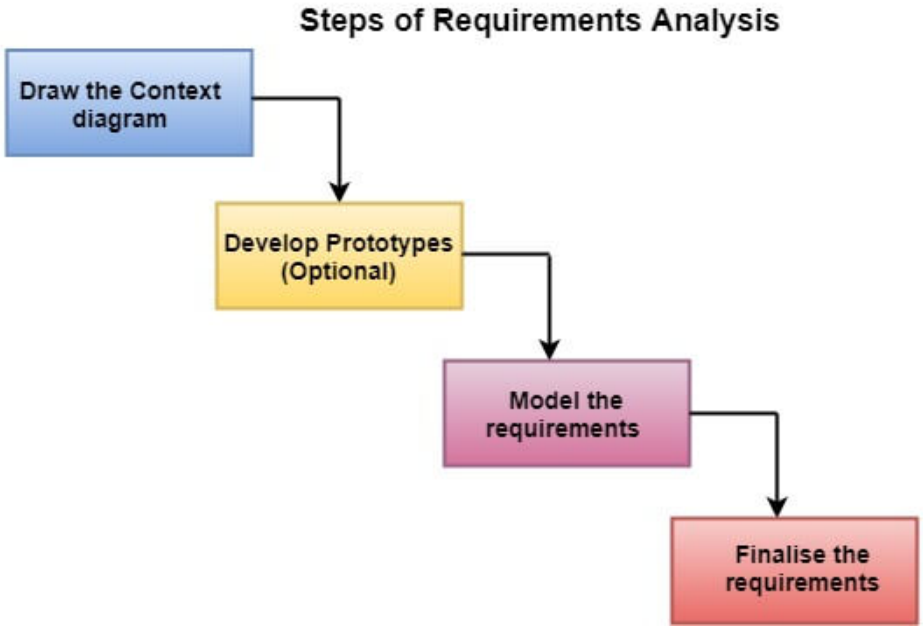

Figure 3: Requirement Analysis

2.3 Design Phase

Description: This includes functional and non-functional, external and internal interfaces, system requirements, use of existing and new frameworks, programming languages, database, reusable modules, and components. For example, if the requirement is to create a new screen to sign-in/sign up users for an app, the different workflows for a new user, existing user, existing user but new mobile, defaulter, VIP user, and so on are detailed in this phase. Even small details — like which fields are mandatory, the type of validations to be placed on fields, etc. — are detailed in this phase.

The designer or architect may create flowcharts, pseudocodes, or algorithms to detail the flow and also list the variable names and descriptions for developers to understand. Similarly, if a call has to be made to an external system for scenarios like authentication, to fetch details, or inserting details into a database, the details of the different tools and systems to be used are mentioned in this phase.

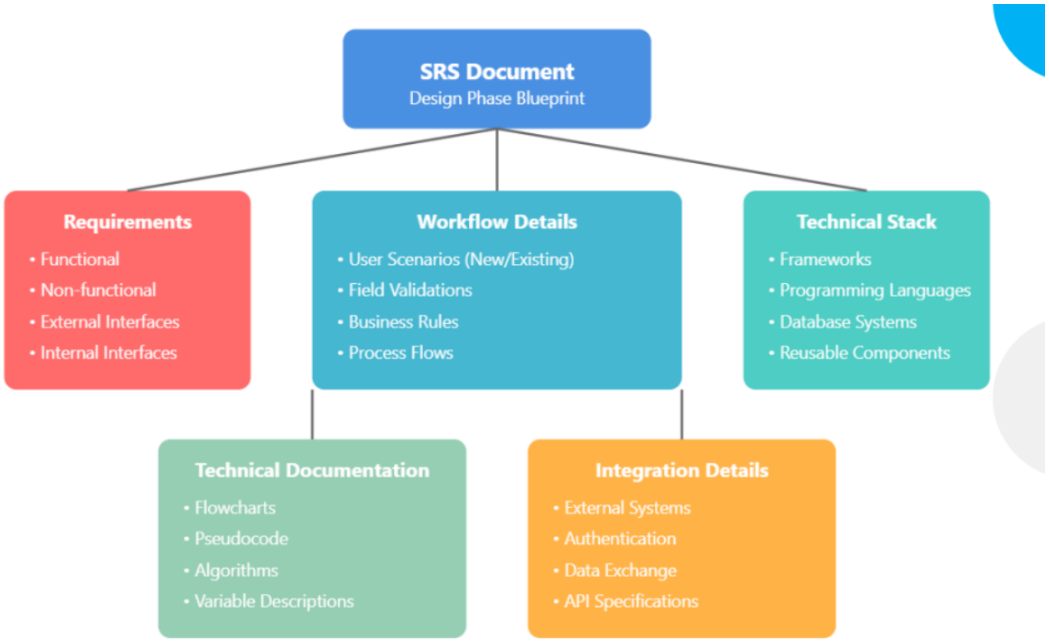

Figure 4: Design Plan

2.4 Coding Phase

Description: The actual work begins in the software development phase, where the development team closely works with the designers and analysts to code and bring to life the design they created. The SRS already contains details only about the overall framework and workflow. The developers create the necessary classes, objects, and handle exceptions. They follow code best practices, create reusable components, and make the code lightweight. They also need to follow the security protocols and ensure high software performance. Usually, developers also conduct unit testing of the code before handing it over to the testing team.

At this stage, the fundamental development of the product starts. For this, developers use a specific programming code as per the design in the DDS. Hence, it is important for the coders to follow the protocols set by the association. Conventional programming tools like compilers, interpreters, debuggers, etc. are also put into use at this stage. Some popular languages like C/C++, Python, Java, etc. are put into use as per the software regulations.

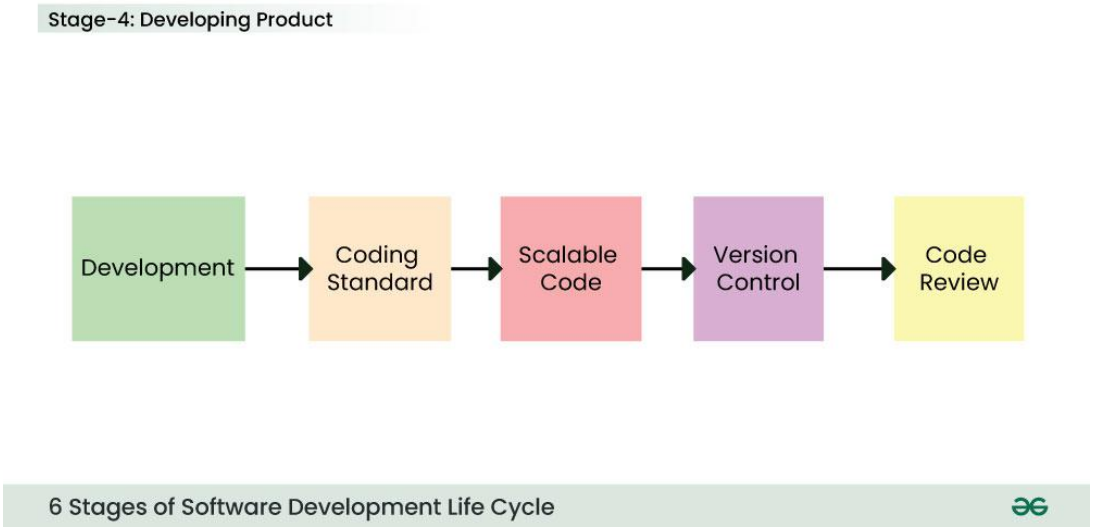

Figure 5: Coding Phase

2.5 Testing Phase

Description: The testing phase includes both functional and non-functional testing. The testing team prepares a framework for testing and writes test cases covering all the project scenarios based on the SRS document, as the software development phase starts. As each module is developed, they run the relevant test cases and mark them as passed or failed. In case a test case does not pass, it is marked as a bug and goes back to the development team for a fix.

Once unit testing is complete, the modules go through the next levels of testing, like integration testing, system testing, performance testing, and security testing. There are many tools that testers use to automate many of the tests, saving time and resources.

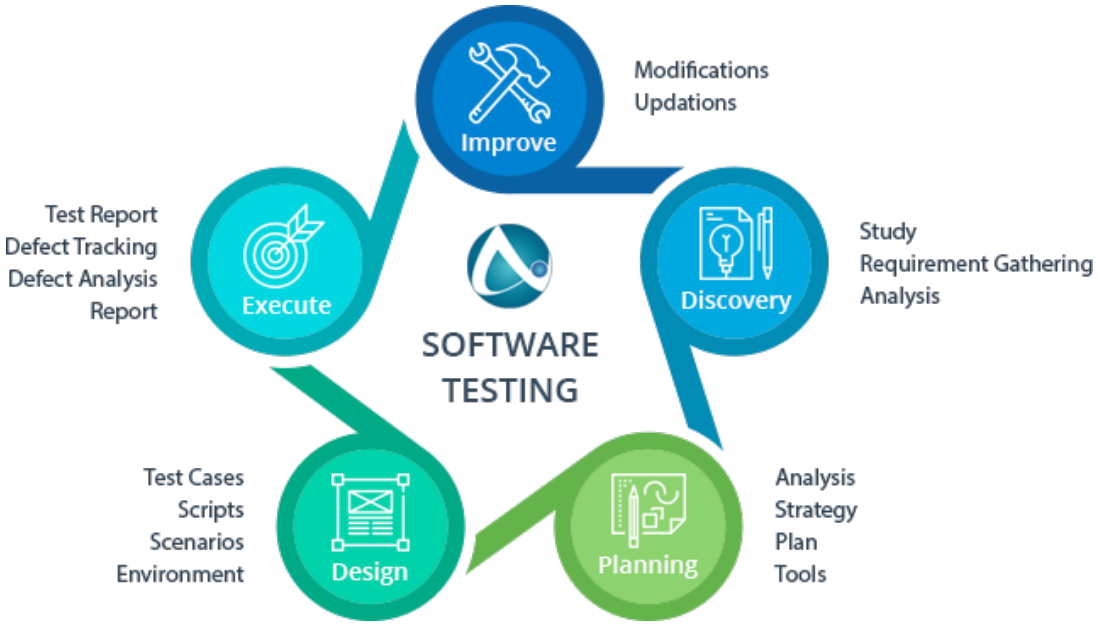

Figure 6: Software Testing

2.6 Deployment and Maintenance Phase

Description: Once testing is done in different stages, the software is ready to be deployed. It is first deployed to a staging or pilot environment, to perform another round of regression, integration, and user acceptance testing (UAT), before finally being put into the production environment. With DevOps and continuous integration and deployment (CI/CD) in place, many projects have automated deployments. Once the software is available to users, they might require assistance or encounter challenges using certain features. Additionally, unexpected bugs may surface, all of which require fixes during the support phase. Some issues may manifest as change requests, initiating another software development lifecycle. Apart from providing support, continuous monitoring, upgrading, and maintenance are essential to ensure the software's seamless functionality.

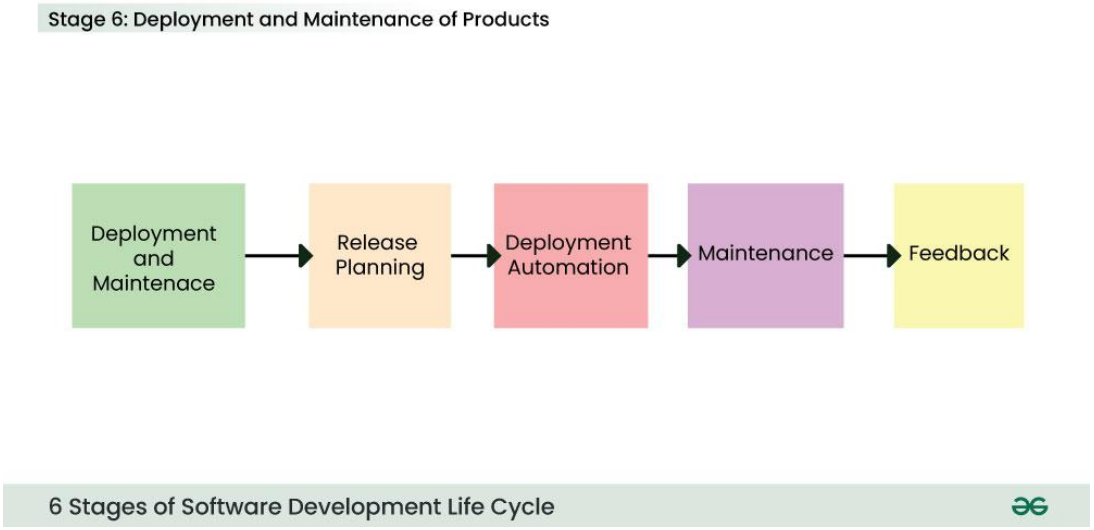

Figure 7: Software Deployment and Maintenance

3 Case Study I

3.1 Introduction

This case study explores the software development process model used in the SIS-ASTROS project, a collaboration between the Federal University of Santa Maria (UFSM) and the Brazilian Army (BA). The project aimed to develop an Integrated Simulation System to support military training in tactical operations related to the use of an ASTROS battery (Artillery Saturation Rocket System).

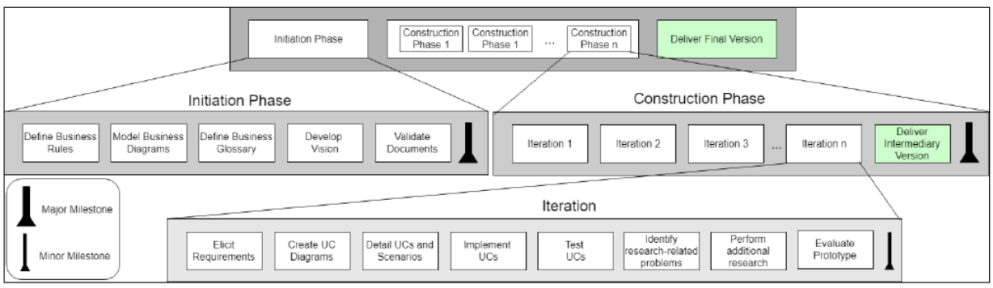

Fig. 1. Life Cycle Vision

Figure 8: Case Study

3.2 Project Context

The SIS-ASTROS project began in 2015 and was expected to conclude in 2020. The primary objective was to create an integrated simulation system for training military personnel in rocket artillery battery operations. The project team included professors, researchers, developers, and students, creating a dynamic yet challenging work environment.

The project faced challenges such as:

- The difficulty in defining system requirements due to the complexity of military operations.
- The unfamiliarity of the development team with military doctrines and terminologies.
- The need for innovative computational solutions, including 3D scenario generation and autonomous navigation.
- High team turnover due to student involvement.

3.3 Software Development Process

A hybrid process model was adopted, incorporating both plan-driven and agile methodologies:

- **Plan-driven elements:** Structured phases, contractual milestones, and formal documentation to meet government requirements.
- **Agile principles:** Incremental development, iterative improvements, and collaborative problem-solving.

The development cycle included two primary phases:

- **Initiation phase:** Defined business rules, modeled domain diagrams, and developed vision documents.
- **Construction phase:** Consisted of iterative cycles involving requirement definition, modeling, implementation, testing, and validation.

3.4 Challenges and Solutions

The project encountered several challenges, which were addressed through the following strategies:

- **Domain Knowledge Gap:** Creation of business diagrams and glossaries to facilitate communication.
- **Requirement Uncertainty:** Iterative development with periodic software version reviews by the client.
- **Complexity of Solutions:** Dedicated research efforts and academic studies to explore necessary technologies.
- **Communication Barriers:** Establishment of a structured communication process and appointment of a Product Owner.
- **High Team Turnover:** Encouraging teamwork, mentorship between senior and junior members, and maintaining documentation.
- **Requirement Changes:** Implementation of a formal change management process.

3.5 Outcomes and Lessons Learned

The hybrid process successfully met project goals while accommodating both the structured needs of the military and the dynamic, research-driven nature of academia. Key lessons learned included:

- The importance of incremental development with frequent client feedback.
- The effectiveness of visual aids (diagrams, prototypes) in understanding complex domains.
- The benefits of a well-defined communication structure in reducing development delays.
- The necessity of an adaptive approach to managing changing requirements.

3.6 Conclusion

The case study demonstrated that a well-balanced hybrid software process model can effectively manage the complexities of university-government collaborations. Future work involves periodic reviews and optimizations of the development process to further enhance efficiency and adaptability.

4 Case Study II

A **Library Management System (LMS)** is a software application designed to manage and streamline library operations, including book tracking, user management, and transaction records. The goal is to create a more efficient and automated system that minimizes manual errors and optimizes resource management.

This case study explores the development of the LMS using the **Software Development Life Cycle (SDLC)** approach.

4.1 Feasibility Study

The feasibility study analyzed the practicality of developing an LMS for a university library. The main factors considered were:

- **Technical Feasibility**: Use of PHP, MySQL, and Bootstrap for a scalable, interactive system.
- **Economic Feasibility**: Cost-benefit analysis confirmed that automating library operations would save time and reduce labor costs.
- **Operational Feasibility**: The system simplifies book management and enhances user experience.

4.2 Requirement Analysis

Requirements were gathered from stakeholders, including librarians, students, and administrators. The functional and non-functional requirements were documented in the Software Requirement Specification (SRS).

Key Functional Requirements:

- User authentication (Admin, Librarian, Students).
- Book search and issue/return management.
- Fine calculation for overdue books.

Key Non-Functional Requirements:

- High system availability and security.
- Scalability for future expansion.

4.3 Design Phase

The design phase included creating system architecture, data flow diagrams, and an entity-relationship model.

ER Model for LMS:

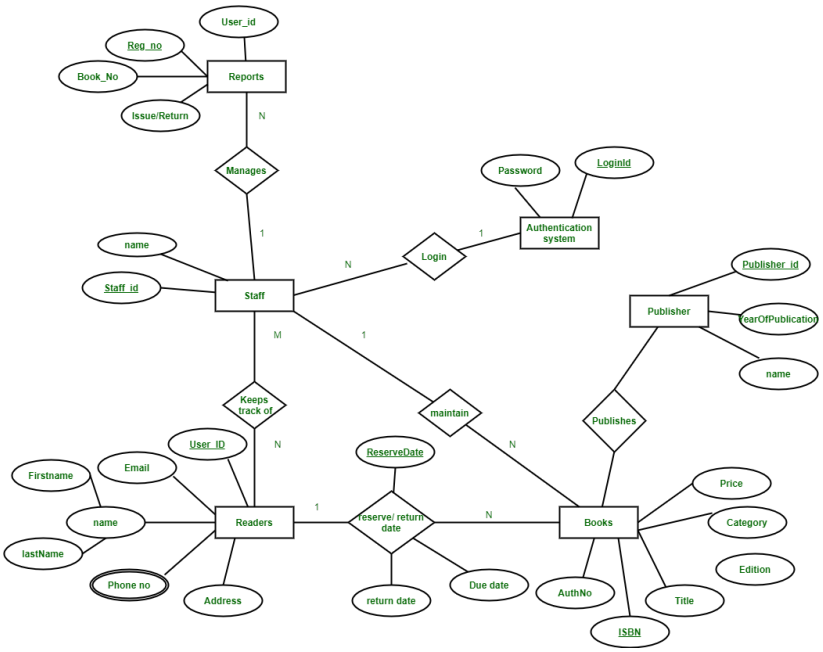

Figure 9: Entity-Relationship Model for LMS

Use Case Diagram:

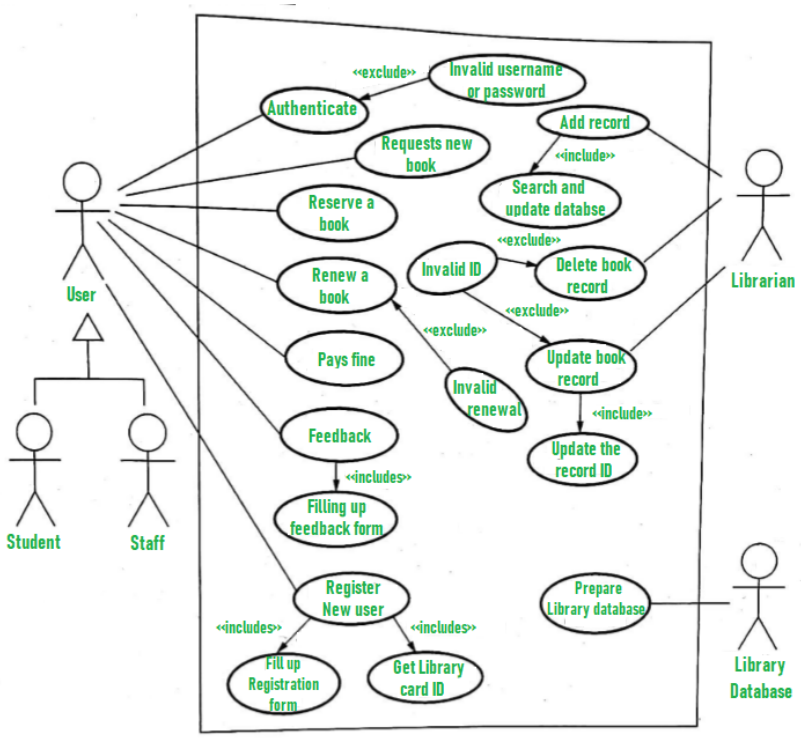

Figure 10: Use Case Diagram for LMS

Data Flow Diagram (DFD):

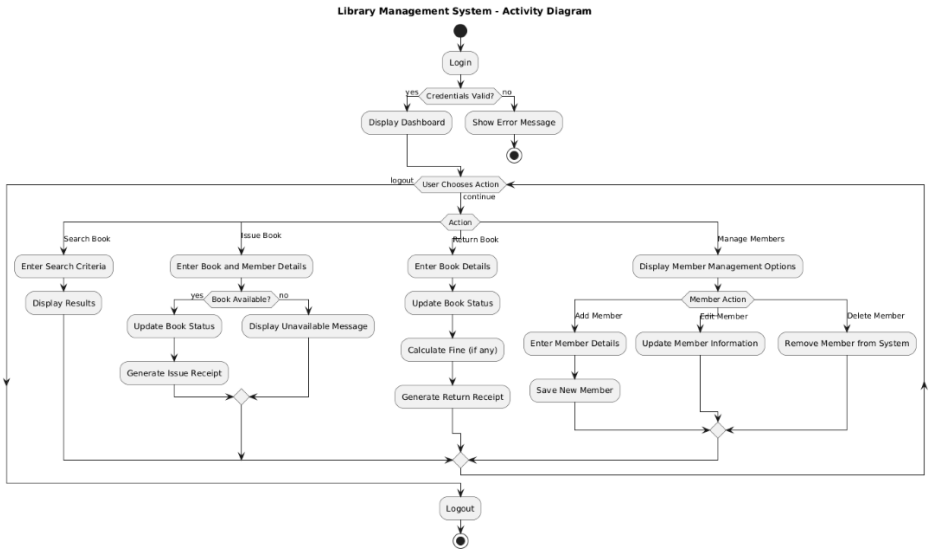

Figure 11: Data Flow Diagram for LMS

4.4 Coding Phase

The LMS was implemented using:

- **Frontend:** HTML, CSS (Bootstrap), JavaScript.
- **Backend:** PHP and MySQL for database management.

Features were developed in modules:

- User authentication system.
- Book search and cataloging.
- Transaction management for book issues and returns.

4.5 Testing Phase

Testing was performed using unit testing, integration testing, and user acceptance testing (UAT). The system was tested for:

- Correct login authentication.
- Smooth book search and checkout processes.
- Proper fine calculation and overdue alerts.

4.6 Deployment and Maintenance Phase

After successful testing, the LMS was deployed on a university server. Future improvements include:

- Mobile app integration.
- AI-based book recommendations.
- Cloud storage for better scalability.

5 References

1. Feasibility Study - Javatpoint
2. Feasibility Study - ResearchGate
3. Software Development Lifecycle - GeeksforGeeks
4. Software Development Lifecycle - MongoDB
5. Software Development Lifecycle - Atlassian
6. Software Development Lifecycle - AWS
7. Software Testing
8. Case Study I
9. Case Study II