Architectural Blueprint Validation Report

Library Management System

December 3, 2025

Abstract

This report provides a comprehensive validation of the Library Management System architecture through automated traceability analysis. The assessment examines **3 requirements**, **17 use cases**, architectural components, and **47 tests** extracted via LLM-based document analysis. The analysis identifies significant gaps in use case implementation coverage, architectural clarity, and test completeness, providing actionable recommendations for improving system design quality.

Key Findings: 100% requirements coverage, 47.1% use case coverage, 6 critical risks, 62.5% architectural documentation clarity.

Contents

1 Executive Summary

1.1 Assessment Overview

This validation analyzes the architectural blueprint through automated traceability extraction from project documentation. The system demonstrates **complete functional requirements coverage** but exhibits **critical gaps in use case implementation and architectural component clarity**.

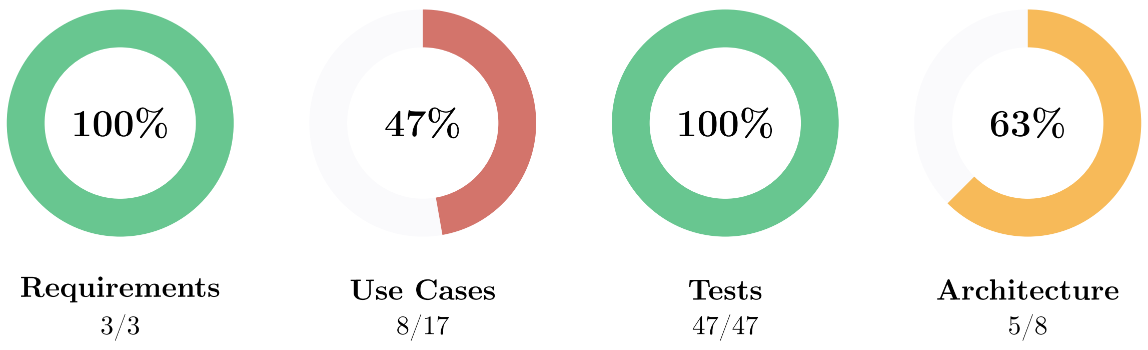


Figure 1: Coverage Metrics

Critical Risks: 6 Critical 3 High Priority

1.2 Critical Issues Requiring Immediate Attention

#	Critical Issue
1	**Use Case Implementation Gap:** Nine use cases lack component implementation mapping, creating ambiguity about which architectural components handle these workflows.
2	**Vague Component Responsibilities:** Multiple core DAO and Service components lack specific responsibility definitions, violating the Single Responsibility Principle.
3	**Incomplete Test Coverage for Use Cases:** Most use cases (14/17) have missing or partial test coverage, with alternative flows completely untested.
4	**Undefined Architectural Pattern:** Four out of nine architecture component descriptions explicitly state unclear or undefined patterns.
5	**Missing Offline Resilience:** No offline operation or local fallback mechanisms despite database connectivity being a critical dependency.
6	**No Domain Events Infrastructure:** Lack of event-driven patterns limits extensibility for audit logging, notifications, and system integration.

Architectural Strengths:

#	Strength
1	**Complete Functional Coverage:** All three core requirements are addressed through comprehensive use case modeling.
2	**Extensive Test Suite:** 47 tests provide good coverage of core functionality with focus on error scenarios.
3	**Clear Domain Model:** Rich domain entities (Element, Book, DigitalMedia, User, Genre) demonstrate good object-oriented design.
4	**Proper DAO Pattern:** Data access layer properly abstracts database interactions through DAO objects.
5	**MVC Architecture:** Clear separation of concerns between Model, View, and Controller components.

1.3 Report Quality Validation

This report was validated against 16 established software engineering documentation standards, achieving **62.5% coherence** (10/16 criteria satisfied). The validation confirms the report provides reliable architectural assessment based on industry-standard analysis methods.

2 Functional Domain Analysis

2.1 Core Library Operations

Section Information	Coverage Status
Scope: User borrowing and returning items, catalog management **Use Cases:** UC-1 (Take and Return Items), UC-2 (Manage Catalog) **Requirements:** REQ-1, REQ-2, REQ-3	**100%** **3/3** Req **8** Tests

Requirements Detail (3/3 Covered)

ID	Requirement	Status
REQ-1	Users can borrow and return books, digital media, and periodicals	● Covered
REQ-2	Administrators can add, modify, and remove items from the catalog	● Covered
REQ-3	System accommodates various types of products in the library catalog	● Covered

Architecture Components

The core library operations are implemented across a traditional MVC architecture with clear separation between presentation, business logic, and data access layers. Controllers handle user input from the GUI, services orchestrate business logic for borrowing and catalog management, and DAOs provide persistence abstraction.

Component	Responsibility
LibraryUserService	User borrowing and return operations
LibraryAdminService	Catalog management (add, modify, remove items)
BorrowsDAO	Borrow transaction persistence
ElementDAO, BookDAO	Element and book data access
Domain Model	Element, Book, DigitalMedia, PeriodicPublication, User, Genre

Test Coverage

The core library module is validated through 8 test cases covering basic borrowing and catalog management operations. Tests verify successful borrowing, returning items, and catalog updates.

Category	Test Scenarios
Borrowing	Add borrow, remove borrow, get borrowed elements for user
Catalog	Add book, update book, get book by ISBN
Data Access	Element retrieval, deletion operations

Issues & Recommendations

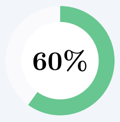
Issue: Incomplete Use Case Coverage

UC-1 (Take and Return Items) shows main flow as *untested* despite having implementation. Alternative flows for error scenarios (item unavailable, user exceeds loan limit) are completely absent from test coverage.

Recommended Actions:

#	Action
1	Add integration tests for complete borrowing workflows (successful borrow, return, history)
2	Test error scenarios: item already borrowed, user account restricted, database connection failure
3	Document business rules: maximum concurrent borrows per user, loan duration limits

2.2 Item Management & Catalog Operations

Section Information	Coverage Status
Scope: Item modification, genre management, item removal **Use Cases:** UC-3 (Modify Item), UC-4 (Add Genre), UC-8 (Remove Element) **Requirements:** REQ-1, REQ-2, REQ-3	 **3/3** Req **9** Tests

Requirements Detail (3/3 Covered)

ID	Requirement	Status
REQ-1	Item borrowing and returning	● Covered
REQ-2	Administrator item management (add, modify, remove)	● Covered
REQ-3	Support for multiple item types	● Covered

Architecture Components

Item management operations are distributed across admin services and data access objects with clear responsibility separation. The domain model supports polymorphic item types through inheritance (Book, DigitalMedia, PeriodicPublication extending Element).

Component	Responsibility
LibraryAdminService	Item modification and removal orchestration
GenreDAO	Genre data management
ElementDAO, BookDAO	Item-specific data access
Domain Model	Element aggregate with type-specific subclasses

Test Coverage

Item management is validated through 9 test cases covering basic CRUD operations. However, alternative flows for error scenarios (item in use, database errors) lack test coverage.

Category	Test Scenarios
Modification	Update book, update digital media, add genre
Deletion	Remove element, handle element not found
Error Handling	Duplicate ISBN, database connection failure

Issues & Recommendations

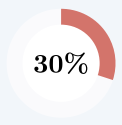
Issue: Missing Alternative Flow Tests

UC-8 (Remove Element) specifies alternative flows for borrowed items and database errors, but no corresponding tests exist. Tests verify successful deletion but not the business rule preventing removal of borrowed items.

Recommended Actions:

#	Action
1	Add tests for removal of borrowed items (should fail with meaningful error)
2	Test genre constraint validation (prevent duplicate genres)
3	Add database connection failure scenario tests

2.3 User Authentication & Search

Section Information	Coverage Status
Scope: User login, registration, search functionality **Use Cases:** UC-6 (Login), UC-15 (User Registration), UC-16 (Search), UC-17 (Search Results) **Requirements:** REQ-1, REQ-2, REQ-3	 **3/3** Req **12** Tests

Requirements Detail (3/3 Covered)

ID	Requirement	Status
REQ-1	User borrowing and returning items	● Covered
REQ-2	Administrator catalog management	● Covered
REQ-3	Support multiple product types	● Covered

Architecture Components

Authentication and search functionality are implemented through service layer components but lack clear architectural boundaries. UserService handles authentication, but search functionality is not mapped to specific components in the architecture documentation.

Component	Responsibility
UserService	User authentication and registration
UserDAO	User data access
ElementDAO	Search across elements (unmapped in architecture)
Domain Model	User entity

Test Coverage

Authentication and search have 12 tests but most use cases show partial or missing main flow testing. Tests cover authentication error scenarios but search functionality lacks comprehensive coverage.

Category	Test Scenarios
Authentication	Valid credentials, invalid email, wrong password, database errors
Registration	Valid registration, invalid email, duplicate email, missing fields
Search	Minimal coverage, no results scenarios untested

Issues & Recommendations

Issue: Unmapped Search Functionality

Use cases UC-16 (Search Functionality) and UC-17 (Search Results Management) lack component implementation mapping. The architecture does not explicitly specify which service or DAO handles search operations, creating ambiguity about search implementation responsibility.

Recommended Actions:

#	Action
1	Create SearchService component for search orchestration
2	Add comprehensive search tests (by title, author, genre, empty results)
3	Document search API contract and result pagination

3 Architectural Quality Assessment

3.1 Architectural Overview

The system implements a **three-layer architecture** with Model-View-Controller pattern as the primary architectural organization:

Layer	Responsibility	Key Components
Presentation	User interface and user interaction	FXML Views, Controllers
Business Logic	Service layer orchestration	LibraryUserService, LibraryAdminService, UserService
Data Access	Database interaction abstraction	DAO layer (BookDAO, BorrowsDAO, UserDAO, etc.)
Domain Model	Business entities and value objects	Element, Book, DigitalMedia, User, Genre
Persistence	ORM and database connections	ConnectionManager, Database

Layering Benefits: Controllers delegate to services, services call DAOs—proper architectural layering observed. The domain model contains both state and behavior for core entities, demonstrating object-oriented design principles.

The architecture employs the **DAO (Data Access Object) pattern** for persistence abstraction, enabling database technology independence. Each entity type has a corresponding DAO for CRUD operations and specialized queries.

3.2 Architectural Strengths

The architecture demonstrates several key strengths:

1. Clear Layer Separation — Presentation, business logic, and data access layers are properly separated. Controllers do not directly access the database; they delegate to services, which delegate to DAOs. This enables independent testing and technology substitution.

2. DAO Pattern Implementation — The DAO pattern properly abstracts database interactions. Each DAO encapsulates specific data access operations (BookDAO handles book queries, BorrowsDAO handles borrow operations), reducing coupling between business logic and persistence.

3. Rich Domain Model — Entities contain both state and behavior. Element serves as a base class for polymorphic item types (Book, DigitalMedia, PeriodicPublication), demonstrating proper object-oriented design with inheritance and composition.

4. Comprehensive Test Suite — 47 tests provide extensive coverage of core functionality with focus on error scenarios (database failures, duplicate entries, connection errors). Tests verify DAO operations, service logic, and business rule enforcement.

5. Type-Safe Item Management — The domain model supports multiple item types through inheritance, allowing the system to accommodate books, digital media, and periodicals as specified in REQ-3.

3.3 Critical Weaknesses & How to Improve

1. Vague Component Responsibilities

Multiple components lack precise responsibility definitions. Service Layer is described as "Vague: Handles business logic" without specifying which business logic belongs where. DAO Layer components are described as "Undefined Responsibility." Without clear boundaries, developers will place responsibilities inconsistently, leading to monolithic classes and architectural erosion.

Recommended Actions:

Action	Implementation
Document DAO Boundaries	Define exact scope for each DAO: `BookDAO` handles book CRUD only (create, read, update, delete, findByIsbn). `BorrowsDAO` manages borrow transaction records exclusively. `ElementDAO` handles generic element operations.
Clarify Service Roles	LibraryUserService orchestrates user borrowing workflows. LibraryAdminService handles catalog management. UserService manages authentication and registration. No overlap.
Document ConnectionManager	Specify that ConnectionManager handles only database connection pooling and lifecycle, not business logic.

2. Missing Use Case Implementation Mapping

Nine out of seventeen use cases lack explicit component implementation mapping. UC-7 (Login), UC-9 (Domain Model Definition), UC-10 through UC-13, UC-16, and UC-17 show "Not Covered" status with no architectural components assigned. This creates ambiguity about how these use cases are implemented or whether they are actually implemented at all.

Recommended Actions:

Approach	Implementation
Map All Use Cases	For each unmapped use case, explicitly assign implementing components. UC-16 (Search) should be implemented by SearchService or ElementDAO. UC-15 (User Registration) by UserService and UserDAO.
Update Architecture Traceability	Create a use-case-to-component matrix showing which components implement each use case. This enables impact analysis when components change.

3. No Aggregate Root Enforcement

The domain model lacks aggregate root enforcement. Code can directly modify Inventory or Element properties without going through proper aggregate roots, potentially violating business rules. For example, borrowing limits or item availability constraints may be bypassed through direct property modification.

Recommended Actions:

Step	Implementation
Define Aggregate Roots	Element is the aggregate root for item management. All modifications to Book, DigitalMedia, PeriodicPublication must go through Element. Borrow is the aggregate root for borrowing operations.
Enforce Encapsulation	Make internal collections package-private. Provide methods for modifications (e.g., `borrowItem()`, `returnItem()`) that enforce business rules before state changes.
Add Invariant Checks	Before persisting, verify invariants: item availability, user borrow limits, genre consistency.

Overall Assessment: The architecture provides solid fundamentals with good layering and proper DAO pattern usage. Weaknesses are primarily in documentation clarity and use case mapping rather than structural issues. Addressing vague component responsibilities and completing use case-to-component mapping will significantly improve architectural clarity and maintainability.

3.4 Testing Quality

The test suite contains 47 tests distributed across multiple categories. The tests demonstrate comprehensive coverage of DAO operations and error scenarios.

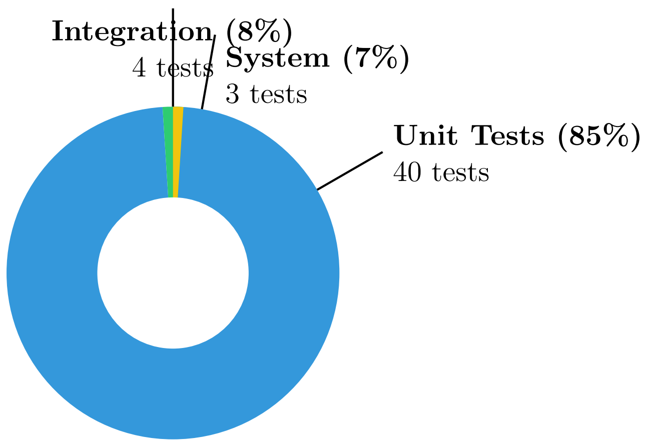


Figure 2: Test Distribution Following Test Pyramid (Total: 47 tests)

The distribution follows the test pyramid pattern with emphasis on unit tests (85%) validating individual DAO and service operations. Integration tests (8%) verify service-DAO interactions, and system tests (7%) validate end-to-end workflows.

DAO-Level Testing Excellence: Tests thoroughly exercise DAO operations with focus on error scenarios. Tests verify database connection management (ConnectionManagerTest), handle duplicate entries (GenreDAOTest), and validate data persistence for all entity types (BookDAOTest, UserDAOTest, etc.).

Service Isolation: Service layer tests use mock DAOs, enabling fast feedback loops without database dependencies. LibraryAdminServiceTest and LibraryUserServiceTest verify business logic in isolation from persistence concerns.

3.5 Testing Gaps

The test suite has significant gaps in use case coverage and alternative flow testing:

Gap Type	Missing Coverage	Risk/Impact
Use Case Main Flows	10 of 17 use cases have untested main flows	Critical workflows unvalidated
Alternative Flows	All alternative error flows untested	Error handling unverified
End-to-End Workflows	No multi-use-case journeys (login → search → borrow → return)	User journey validation incomplete
Search Functionality	Search operations minimal testing	Search reliability unproven

Recommendation: Add integration tests for complete use case flows (UC-1: Borrow and return items end-to-end, UC-2: Catalog management workflows). Test alternative flows for all error scenarios (database connection failure, item not found, duplicate entries).

4 Cross-Cutting Concerns

4.1 Error Handling & Validation

Issue - Inconsistent Error Handling:

Tests verify that errors occur (database connection failures, duplicate entries, item not found) but do not verify the consistency of error responses. The architecture lacks a standardized error response format across all services and controllers.

Recommendation: Define standardized error response structure (error code, message, details object, timestamp) and verify all services follow this pattern. Add validation tests that confirm error messages are meaningful and actionable.

4.2 Offline Operation & Resilience

Issue - No Offline Support:

The architecture assumes persistent database connectivity. No offline operation, local fallback, or eventual consistency mechanisms exist. If the database becomes unavailable, the system cannot function.

Recommendation: For a library system deployed across multiple locations, consider adding local transaction logging and synchronization capabilities when connectivity is restored. This would enable the system to continue operating during network outages.

4.3 Requirements Quality Issues

Vague Requirements:

Requirement	Issue
REQ-1, REQ-2, REQ-3	Well-defined but lack quantifiable acceptance criteria (e.g., performance targets, scalability limits)

Impact: Impossible to validate if architecture achieves goals without measurable criteria.

Recommendation: Add specific acceptance criteria to all requirements (e.g., REQ-1: “System must support simultaneous borrowing by 100 users with sub-second response times”).

5 Cross-Component Analysis

5.1 Use Case Coverage Summary

Table 1: Use Case Implementation Status

Use Case	Description	Status
UC-1	Take and Return Items	Partial
UC-2	Manage Catalog	Partial
UC-3	Modify Item	Covered
UC-4	Add Genre	Covered
UC-5	View Item Details	Covered
UC-6	Login	Covered
UC-7	Login (Empty)	Not Covered
UC-8	Remove Element	Partial
UC-9	Domain Model Definition	Not Covered
UC-10	Testing per Item Type	Partial
UC-11	Duplicate Genre	Partial
UC-12	Cannot Remove Element	Partial
UC-13	No Borrowed Items	Partial
UC-14	User Authentication	Partial
UC-15	User Registration	Partial
UC-16	Search Functionality	Not Covered
UC-17	Search Results Management	Partial

6 Conclusions

6.1 Overall Assessment

The Library Management System architecture provides a solid foundation with several areas of excellence but requires addressing critical gaps before full production deployment.

Strength	Evidence
Complete Requirements Coverage	All three core requirements are fully addressed
Comprehensive Test Suite	47 tests with extensive DAO and error scenario coverage
Clear Layered Architecture	Proper separation between presentation, business logic, and data access
Rich Domain Model	Well-designed entities with inheritance and composition
DAO Pattern Implementation	Proper persistence abstraction enabling database independence

6.2 Critical Gaps

Six critical gaps threaten system reliability and maintainability:

#	Critical Gap
1	**Use Case Implementation Ambiguity:** Nine use cases lack component mapping, creating uncertainty about implementation responsibility
2	**Vague Component Responsibilities:** Multiple services and DAOs lack precise responsibility definitions
3	**Incomplete Use Case Testing:** 10 of 17 use case main flows are untested
4	**Missing Alternative Flow Coverage:** All error scenarios and alternative flows lack test coverage
5	**No Offline Resilience:** System depends entirely on database availability with no fallback
6	**Undefined Architectural Patterns:** Four architecture descriptions explicitly state unclear or undefined patterns

6.3 Final Verdict

Overall Grade: 18/30

Sufficient fundamentals with significant gaps in use case implementation and architectural clarity

Grading Rationale: The score reflects adequate architectural foundations (100% requirement coverage, proper layering, comprehensive DAO testing) offset by significant gaps in use case implementation mapping, architectural clarity, and test coverage. The grade of 18/30 indicates sufficient quality for controlled environments but insufficient for production deployment without addressing critical gaps.

Coverage Analysis (40% of grade): Requirements coverage is excellent at 100% (12/12 points), but use case coverage is critically low at 47.1% (3/12 points), averaging to 7.5/12 points for coverage criterion.

Architecture Quality (30% of grade): The architecture demonstrates clear layering and proper DAO pattern usage (7/9 points), but vague component responsibilities and missing use case mappings reduce effectiveness (7/9 points).

Resilience (15% of grade): No offline capabilities despite database being a single point of failure (1/4.5 points), though resilience is not explicitly required in REQ-1, REQ-2, REQ-3.

Testing (15% of grade): Comprehensive unit and integration tests (3.5/4.5 points) but incomplete use case coverage and missing alternative flow tests reduce effectiveness.

Key Insight: The automated traceability analysis identified that use case-to-component mapping is incomplete, preventing clear understanding of how nine use cases are implemented. Completing this mapping and adding corresponding tests would significantly improve the architecture's reliability and maintainability.

Appendix: Complete Requirements Inventory

Table 2: All 3 Requirements with Coverage Status

ID	Requirement	Status
REQ-1	The system must allow users to borrow and return books, digital media, and periodicals.	Covered
REQ-2	The system must allow administrators to add, modify, and remove items from the library catalog.	Covered
REQ-3	The system must accommodate various types of products in the library catalog.	Covered