Software Engineering Report
Validation Analysis

Automated Validation System

December 2, 2025

Contents

1 Executive Summary

1.1 Validation Status

Table 1: Overall Validation Metrics

Metric	Value
Total Requirements Extracted	18
Total Use Cases Extracted	6 (6 explicit, 0 implicit)
Total Architecture Components	15
Total Tests Extracted	4 logical test suites (51 individual tests reported)
Requirements Coverage	100% (18/18 mapped to at least one use case or component)
Use Case Coverage	100% (6/6 mapped to architecture and tests)
Test Coverage	100% of described use cases have at least partial test coverage
Feature-Based Coverage	82.0% (41/50 reference features covered – approximate)
Overall Status	**PASSED (with minor documentation gaps)**

1.2 Key Findings

- **Strengths:**
  - Clear and coherent description of actors, responsibilities and main use cases.
  - Architecture is well-structured into Business Logic, Domain Model and ORM packages, with good separation of concerns.
  - Systematic unit testing strategy for controllers and DAO classes, with 51 tests executed successfully.
  - Consistent use of design patterns (Strategy, Singleton) and layered persistence design.
- **Areas for Improvement:**
  - Functional and non-functional requirements are implicit in narrative text; there is no dedicated requirements section with identifiers.
  - Non-functional requirements (performance, security, usability) are largely undocumented.
  - Test descriptions are high-level; explicit mapping from individual test methods to specific use case flows is not documented.
  - Error handling and failure scenarios are only partially described in use cases and tests.

2 Content Extraction Results

2.1 Phase 1: Table of Contents and Sections

2.1.1 Extracted Table of Contents

Table 2: Extracted Table of Contents with Page Ranges

Section	**Start Page**	**End Page**
1 Introduzione	3	4
1.1 Statement	3	3
1.2 Tecnologie e Strumenti Utilizzati	3	4
2 Progettazione	5	11
2.1 Use Case Diagram	5	6
2.2 Use case templates	6	8
2.3 Mockup dell'interfaccia utente	8	11
3 UML e Struttura	12	20
3.1 Package Diagram	12	12
3.2 Class Diagram	12	19
3.2.1 Business Logic	12	13
3.2.1.1 Admin	13	13
3.2.1.2 TrainerController	13	13
3.2.1.3 CompanyController	13	13
3.2.1.4 Notifier	13	13
3.2.2 Domain Model	13	16
3.2.2.1 Subscription	15	15
3.2.2.2 FeeStrategy, SingleEmployeeFee e MultipleEmployeeFee	15	15
3.2.2.3 Employee	15	15
3.2.2.4 Company	15	15
3.2.2.5 Trainer	15	15
3.2.2.6 Workshift	15	15
3.2.2.7 Material, Slide e Video	15	16
3.2.2.8 Course	16	16
3.2.2.9 FocusCourse	16	16
3.2.3 Object-Relational Mapping (ORM)	16	19
3.2.3.1 ConnectionManager	16	16
3.2.3.2 SubscriptionDAO	16	16
3.2.3.3 WorkshiftDAO	16	17
3.2.3.4 CompanyDAO	17	17
3.2.3.5 TrainerDAO	17	17
3.2.3.6 MaterialDAO	17	17
3.2.3.7 EmployeeDAO	17	18
3.2.3.8 CourseDAO	18	19
3.3 Database	19	20
4 Testing	21	25
4.1 Test package Controllers	21	22
4.1.1 AdminTest	21	21
4.1.2 CompanyControllerTest	21	22
4.1.3 TrainerControllerTest	22	22
4.2 Test package ORM	22	23
4.2.1 TrainerDAOTest	22	23
4.3 Risultati dei test	25	25

2.1.2 Section Classification

Table 3: Section Classification by Content Type

Section	Labels
1.1 Statement	Requirements, Use Cases
1.2 Tecnologie e Strumenti Utilizzati	[] (context / technologies)
2.1 Use Case Diagram	Use Cases
2.2 Use case templates	Use Cases
2.3 Mockup dell'interfaccia utente	Mockups, Use Cases
3.1 Package Diagram	Architecture
3.2 Class Diagram (and all 3.2.x subsections)	Architecture
3.3 Database	Architecture
4 Testing (and all 4.x subsections)	Test

2.2 Requirements Extraction

From the narrative description (mainly Section 1.1 and the detailed class descriptions), 18 atomic requirements were extracted.

2.2.1 Requirements Quality Distribution

Table 4: Requirements Quality Assessment

Quality Level	Count	Percentage
Well-defined	10	55.6%
Needs Detail	6	33.3%
Vague/Unquantified	2	11.1%
Total	**18**	**100%**

2.2.2 Requirements by Type

Table 5: Requirements Classification

Type	Count	Percentage
Functional	14	77.8%
Non-Functional	2	11.1%
Constraint	1	5.6%
Goal/Background	1	5.6%
Total	**18**	**100%**

2.2.3 Detailed Requirements List

Table 6: Extracted Requirements

ID	Type	Description	Quality
REQ-1	Functional	The system must allow administrators to create training courses by specifying date, time and description.	Well-defined
REQ-2	Functional	The system must allow administrators to modify existing training courses.	Well-defined
REQ-3	Functional	The system must allow administrators to delete existing training courses.	Well-defined
REQ-4	Functional	The system must automatically notify companies and trainers whenever a course is created, modified or deleted.	Needs Detail
REQ-5	Functional	The system must allow administrators to create, modify and delete trainers' workshifts and to view scheduled workshifts.	Needs Detail
REQ-6	Functional	The system must allow administrators and trainers to view the list of employees registered to courses and their personal and work data.	Needs Detail
REQ-7	Functional	The system must allow administrators to send payment reminders to companies for unpaid subscription fees.	Needs Detail
REQ-8	Functional	The system must allow trainers to view their workshifts and assigned courses.	Well-defined
REQ-9	Functional	The system must allow trainers to upload, view and delete multimedia teaching materials (slides and videos) associated with courses.	Well-defined
REQ-10	Functional	The system must automatically notify companies when new teaching materials are uploaded or modified.	Needs Detail
REQ-11	Functional	The system must allow companies to register their employees to courses by providing required employee data and number of lessons.	Well-defined
REQ-12	Functional	The system must allow companies to view the training calendar and multimedia materials uploaded by trainers.	Well-defined
REQ-13	Functional	The system must allow companies to view outstanding subscription fees and pay them through the application.	Needs Detail
REQ-14	Functional	The system must automatically calculate the subscription fee per employee, applying a discount when multiple employees from the same company are registered.	Well-defined

Continued on next page

Table 6 – continued from previous page

ID	Type	Description	Quality
REQ-15	Non-functional	The system must centralize and standardize the sending of notification emails to companies and trainers using a dedicated notification component.	Needs Detail
REQ-16	Constraint	The system must use a relational database (PostgreSQL) to persist domain entities and must access it via JDBC.	Well-defined
REQ-17	Non-functional	The system must be modular, separating business logic, domain model and persistence (ORM) into distinct packages.	Well-defined
REQ-18	Goal/background	The system must support the management of an IT training center, providing advanced functionalities for organizing courses, managing subscriptions and viewing employee data.	Vague/Unquantified

Notes on Requirement Quality

- Requirements REQ-4, REQ-5, REQ-6, REQ-7, REQ-10, REQ-13 and REQ-15 would benefit from more precise conditions (e.g., notification timing, error handling, payment methods).
- Non-functional aspects such as performance, security and usability are not explicitly quantified.

2.3 Use Case Extraction

A total of **6 explicit use cases** are documented with templates (UC-1 to UC-6). No additional implicit use cases were inferred beyond these, as the main flows are already covered.

2.3.1 Use Case Summary

Table 7: Extracted Use Cases

ID	Name	Type	Actors
UC-1	Set Course	Explicit	Admin, System
UC-2	Delete Course	Explicit	Admin, System
UC-3	View Employee List	Explicit	Admin, Trainer, System
UC-4	Upload Videos and Slides	Explicit	Trainer, System, Company (as notified party)
UC-5	Pay Fee	Explicit	Company, System
UC-6	Register Employee	Explicit	Company, System

2.3.2 Detailed Use Case Descriptions

UC-1: Set Course

- **Actors:** Admin, System
- **Type:** Explicit
- **Main Flow:**
  1. User (Admin) accesses the CREATE COURSE page.
  2. System shows a form with fields: Date, Time, Description and Focus.
  3. User fills in the fields and presses the CREATE button.
  4. If all values are correct, the system records the course and notifies Companies and Trainers of the new lesson.
- **Alternative Flows:**
  - If the entered values do not have the required format or are invalid, the system signals the error to the user.

UC-2: Delete Course

- **Actors:** Admin, System
- **Type:** Explicit
- **Main Flow:**
  1. User accesses the DELETE COURSE page.
  2. System shows a list of scheduled courses.
  3. User selects the course to delete and confirms via DELETE SELECTED COURSE.
  4. System deletes the course and notifies Companies and Trainers of the deletion.
- **Alternative Flows:** Not explicitly described.

UC-3: View Employee List

- **Actors:** Admin, Trainer, System
- **Type:** Explicit
- **Main Flow:**
  1. User selects VIEW EMPLOYEES LIST.
  2. System shows a list of employees registered to training courses.
- **Alternative Flows:** Not described.

UC-4: Upload Videos and Slides

- **Actors:** Trainer, System, Company (as notified party)
- **Type:** Explicit
- **Main Flow:**
  1. User selects UPLOAD VIDEO AND SLIDES from the sidebar.
  2. System shows two file pickers, one for each material type.

3. User selects files from their PC and uploads them via the dedicated button.
4. If upload succeeds, companies are informed of the presence of new content.

- **Alternative Flows:**
  - If the upload fails, the user is notified.

UC-5: Pay Fee

- **Actors:** Company, System
- **Type:** Explicit
- **Main Flow:**
  1. User selects PAY FEE in the side menu and views the list of their employees registered to courses, with invoice status.
  2. Selecting an employee with unpaid balance, the user proceeds to payment via the dedicated button.
  3. The fee per meeting is reduced as a function of the number of registered employees; it is full only if the company has a single participant.
  4. System automatically calculates the total amount to be paid.
- **Alternative Flows:** Not described.

UC-6: Register Employee

- **Actors:** Company, System
- **Type:** Explicit
- **Main Flow:**
  1. User selects REGISTER EMPLOYEE from the sidebar.
  2. System shows a form to enter employee information and number of lessons.
  3. By pressing GO!, the registration is performed.
- **Alternative Flows:**
  - If the form is not correctly filled, the user is notified.

2.4 Architecture Extraction

2.4.1 Architectural Pattern

The architecture is explicitly organized into three main packages: Business Logic, Domain Model and Object-Relational Mapping (ORM). This corresponds to a **Layered Architecture** with clear separation between presentation/business logic and persistence.

2.4.2 Architecture Components

Table 8: Architecture Components Analysis

Component	**Responsibility**	**Design Notes**
Business Logic package	Implements system use cases and coordinates interactions between domain entities and persistence.	Clear separation of concerns; controllers per actor plus a dedicated Notifier component.
Admin (controller)	Manages creation, modification, deletion and viewing of courses; manages workshifts; views employees and sends payment reminders.	Single class with several responsibilities related to admin actor; cohesive around "administration" but somewhat broad.
TrainerController	Provides trainer operations: view workshifts and courses, view employees and their info, upload/delete/view materials.	Focused on trainer-related operations; interacts with Domain Model and DAO layer.
CompanyController	Provides company operations: register employees, pay fees, view courses, view materials, view employee info.	Cohesive around company actor; encapsulates fee strategy selection logic via DAO.
Notifier	Singleton component that sends HTML emails to companies and trainers via SMTP (Gmail).	Good centralization of notification logic; explicit use of Singleton pattern.
Domain Model package	Contains core domain entities and value types independent of persistence and controllers.	Clean domain-centric design; uses Strategy and enum types.
Subscription	Represents an employee's subscription, including number of meetings, fee paid flag, associated Employee and FeeStrategy.	Encapsulates subscription state; depends on FeeStrategy for fee calculation.
FeeStrategy / SingleEmployeeFee / MultipleEmployeeFee	Strategy pattern for computing subscription fee depending on whether a single or multiple employees are registered.	Good use of Strategy; supports extension and encapsulates discount logic.
Employee	Represents an employee with personal data, associated Subscription and Company.	Standard entity; relationships to Company and Subscription are explicit.
Company	Represents a company with identification data and list of employees.	Aggregates employees; used by controllers and DAO.
Trainer	Represents a trainer with personal data and list of Workshifts.	Simple entity; supports scheduling.

Continued on next page

Table 8 – continued from previous page

Component	Responsibility	Design Notes
Workshift	Represents a workshift with date and start time.	Used for scheduling trainers and courses.
Material (abstract), Slide, Video	Abstract multimedia content with upload metadata and trainer reference; Slide and Video specialize it.	Good abstraction for different media types; supports extension.
Course	Represents a course with date, time, description and FocusCourse category.	Central entity for scheduling and catalog.
FocusCourse (enum)	Enumerates allowed course categories.	Simple, type-safe categorization.
ORM package	Contains DAO classes and ConnectionManager to handle persistence via JDBC.	Clear persistence layer; DAOs per entity; uses Singleton for connection.
ConnectionManager	Manages a single JDBC connection to PostgreSQL using Singleton pattern.	Optimizes resource usage; centralizes connection handling.
SubscriptionDAO	CRUD and query operations for Subscription, including fee strategy updates and unpaid company retrieval.	Encapsulates fee-related queries; supports business logic in controllers.
WorkshiftDAO	Manages Workshift entities and mapping between trainers and workshifts.	Provides various retrieval methods (all, per trainer, per date).
CompanyDAO	Manages Company persistence and retrieval.	Standard DAO; supports listing and lookup by id.
TrainerDAO	Manages Trainer persistence and retrieval, including shifts and lookup by email.	Used in TrainerDAOTest; supports authentication-like queries.
MaterialDAO	Manages multimedia content persistence, including upload by filename and retrieval by filename.	Uses filename as key; abstracts file storage details.
EmployeeDAO	Manages Employee persistence and retrieval, including queries by company and mapping to Company and Subscription.	Complex mapping logic; reconstructs Subscription and FeeStrategy from DB.
CourseDAO	Manages Course persistence and retrieval, including mapping to FocusCourse and lookup by date/time.	Encapsulates course-related queries and mapping to enum.

Continued on next page

Table 8 – continued from previous page

Component	Responsibility	Design Notes
Database (PostgreSQL)	Relational database storing all domain entities and relationships.	ER and logical schema documented; supports integrity constraints.

2.4.3 Architecture Analysis

Summary: The architecture follows a clear layered pattern (Business Logic, Domain Model, ORM, Database) with good separation of concerns. Controllers encapsulate actor-specific operations, the domain model is clean and pattern-based, and DAOs isolate persistence logic.

Strengths:

- Clear package-level separation between business logic, domain entities and persistence.
- Appropriate use of design patterns (Strategy for fees, Singleton for connection and notifications).
- DAOs encapsulate SQL and mapping logic, keeping domain classes persistence-agnostic.

Observations and Minor Gaps:

- Some controller classes (e.g., Admin) aggregate many responsibilities; further decomposition into service classes could improve cohesion.
- Error handling and transaction management strategies in the persistence layer are not described in detail.
- No explicit description of presentation/UI layer implementation beyond mockups.

2.5 Test Extraction

The Testing section describes test organization and several representative test classes. Individual test method names are not fully listed, but logical test suites and their purposes are described.

2.5.1 Test Type Distribution

Table 9: Test Distribution by Type (Logical Suites)

Test Type	Count (Suites)	Notes
Unit (Controllers)	3	AdminTest, CompanyControllerTest, TrainerControllerTest
Unit (DAO / ORM)	1+	TrainerDAOTest (others implied but not detailed)
Integration/System	0	Not explicitly distinguished from unit tests
Performance	0	Not mentioned
Total Suites	**4+**	51 individual tests reported overall

2.5.2 Detailed Test List (Logical)

Table 10: Extracted Test Suites and Representative Tests

ID	Type	Artifact	Coverage Hint
TEST-1	Unit	AdminTest (class)	Verifies creation, modification, deletion and viewing of courses; verifies viewing of employees list.
TEST-2	Unit	CompanyControllerTest (class)	Verifies employee registration, fee strategy assignment, viewing of materials, viewing of courses and employee info.
TEST-3	Unit	TrainerControllerTest (class)	Verifies upload, viewing and deletion of teaching materials; verifies viewing of employees, courses and workshifts.
TEST-4	Unit	TrainerDAOTest (class)	Verifies insertion, retrieval (by email, all trainers, trainers' shifts) and deletion of trainers.

2.5.3 Test Results

- The report states that all **51 tests** across ORM and Controllers packages completed successfully.
- Tests are ordered using JUnit's `@TestMethodOrder` to ensure logical sequences for DAO operations (insert, retrieve, update, delete).
- Each test includes cleanup logic to maintain independence and repeatability.

3 Consolidated Architecture Model

3.1 Overall Pattern and Layers

- **Pattern:** Layered Architecture (3 main logical layers plus infrastructure)

Table 11: Architectural Layers

Layer	Description
Presentation Layer (implicit)	User interface layer represented by mockups; not implemented in detail in the report but assumed to consist of web pages or GUI screens corresponding to use cases (e.g., CREATE COURSE, PAY FEE).
Business Logic Layer	Contains controllers (Admin, TrainerController, CompanyController) and Notifier; implements use cases and orchestrates domain and persistence operations.

Continued on next page

Table 11 – continued from previous page

Layer	Description
Domain Layer	Contains core entities (Subscription, Employee, Company, Trainer, Workshift, Material/Slide/Video, Course, FocusCourse, FeeStrategy and its implementations).
Persistence/DAO Layer	Contains ConnectionManager and DAO classes (SubscriptionDAO, WorkshiftDAO, CompanyDAO, TrainerDAO, MaterialDAO, EmployeeDAO, CourseDAO) that handle CRUD and queries via JDBC.
Infrastructure Layer	PostgreSQL database and SMTP email infrastructure (Gmail) used by ConnectionManager and Notifier.

3.2 Consolidated Components

Table 12: Consolidated Components by Layer

Component	Responsibility	Layer	Communicates With
UI Screens (mockups)	Provide user interaction for each main use case (course creation/deletion, employee registration, fee payment, material upload, list views).	Presentation	Controllers (conceptually)
Admin	Implements admin use cases: setCourse, modifyCourse, deleteCourse, viewWorkshifts, create/modify/delete workshifts, viewEmployeesList, viewEmployeeInfo, paymentReminder.	Business Logic	Domain entities, DAOs, Notifier
TrainerController	Implements trainer use cases: viewWorkshifts, viewCourses, viewEmployeesList, viewEmployeesInfo, uploadSlideVideo, deleteSlideVideo, viewSlideVideo.	Business Logic	Domain entities, DAOs, Notifier
CompanyController	Implements company use cases: registerEmployee, payFee, viewCourses, viewSlideVideo, viewEmployeeInfo.	Business Logic	Domain entities, DAOs, Notifier
Notifier	Sends HTML emails to companies and trainers via SMTP.	Business Logic / Infrastructure	Controllers, SMTP server

Continued on next page

Table 12 – continued from previous page

Component	Responsibility	Layer	Communicates With
Subscription	Represents subscription details and fee strategy for an employee.	Domain	Employee, FeeStrategy
FeeStrategy, SingleEmployeeFee, MultipleEmployeeFee	Compute subscription fee depending on number of employees from same company.	Domain	Subscription
Employee	Represents an employee with personal data and associated subscription and company.	Domain	Company, Subscription
Company	Represents a company and its employees.	Domain	Employee
Trainer	Represents a trainer and their workshifts.	Domain	Workshift
Workshift	Represents a scheduled workshift.	Domain	Trainer, Course (implicitly)
Material, Slide, Video	Represent multimedia teaching materials uploaded by trainers.	Domain	Trainer
Course	Represents a course with schedule and description.	Domain	FocusCourse, Workshift (implicitly)
FocusCourse	Enumerates course categories.	Domain	Course
ConnectionManager	Manages JDBC connection to PostgreSQL.	Persistence	DAOs, PostgreSQL
SubscriptionDAO	Persists and retrieves Subscription data, including fee strategy and payment status.	Persistence	ConnectionManager, Subscription, Company
WorkshiftDAO	Persists and retrieves Workshift data and trainer-workshift mappings.	Persistence	ConnectionManager, Workshift, Trainer
CompanyDAO	Persists and retrieves Company data.	Persistence	ConnectionManager, Company
TrainerDAO	Persists and retrieves Trainer data and their shifts.	Persistence	ConnectionManager, Trainer, Workshift
MaterialDAO	Persists and retrieves Material (Slide/Video) data.	Persistence	ConnectionManager, Material

Continued on next page

Table 12 – continued from previous page

Component	Responsibility	Layer	Communicates With
EmployeeDAO	Persists and retrieves Employee data, including associated Subscription and Company.	Persistence	ConnectionManager, Employee, Subscription, Company
CourseDAO	Persists and retrieves Course data and FocusCourse mapping.	Persistence	ConnectionManager, Course, FocusCourse
PostgreSQL Database	Stores all domain entities and relationships.	Infrastructure	ConnectionManager
SMTP/Gmail	Sends notification emails.	Infrastructure	Notifier

3.3 Architecture Consolidation Summary

Analysis Summary: The consolidated architecture is coherent and consistent with a typical three-layered enterprise Java application. Responsibilities are mostly well-separated, and the use of DAOs and domain entities is clear. The main missing piece is an explicit description of the presentation technology and error-handling strategies.

4 Traceability Matrix

4.1 Requirements to Use Cases Mapping

Table 13: Requirements to Use Cases Traceability

Req ID	Use Cases	Status	Rationale
REQ-1	UC-1	Covered	UC-1 (Set Course) describes course creation with date, time and description.
REQ-2	UC-1	Covered	Modify course is mentioned in Admin class but not in a dedicated use case; UC-1 partially covers course definition.
REQ-3	UC-2	Covered	UC-2 (Delete Course) describes deletion of a selected course.
REQ-4	UC-1, UC-2	Covered	Notifications on course creation/deletion are mentioned in UC-1 and UC-2; modification notifications are only described in text.
REQ-5	-	Covered	Workshift management is described in Admin class but no explicit use case template is provided.

Continued on next page

Table 13 – continued from previous page

Req ID	Use Cases	Status	Rationale
REQ-6	UC-3	Covered	UC-3 (View Employee List) covers viewing employees and their data by Admin and Trainer.
REQ-7	-	Covered	Payment reminders are described in Admin class but no explicit use case template exists.
REQ-8	-	Covered	TrainerController methods viewWorkshifts and viewCourses are described but not formalized as separate use cases.
REQ-9	UC-4	Covered	UC-4 (Upload Videos and Slides) covers upload and notification; delete and view are described in TrainerController.
REQ-10	UC-4	Covered	UC-4 mentions notification to companies on successful upload; other material changes are not fully specified.
REQ-11	UC-6	Covered	UC-6 (Register Employee) describes employee registration by company.
REQ-12	UC-5, UC-4	Covered	UC-5 shows fee-related view; UC-4 and narrative describe viewing materials and calendar.
REQ-13	UC-5	Covered	UC-5 (Pay Fee) describes viewing unpaid fees and paying them.
REQ-14	UC-5	Covered	UC-5 explicitly states fee reduction based on number of employees and automatic total calculation.
REQ-15	UC-1, UC-2, UC-4	Covered	Use cases mention notifications; Notifier class centralizes them, but no dedicated use case.
REQ-16	-	Covered	Use of PostgreSQL and JDBC is an architectural constraint, not tied to a specific use case.
REQ-17	-	Covered	Package structure (Business Logic, Domain Model, ORM) implements this requirement.
REQ-18	UC-1–UC-6	Covered	All main use cases collectively realize the high-level goal of managing an IT training center.

4.2 Use Cases to Architecture Mapping

Table 14: Use Cases to Architecture Traceability

UC ID	UC Name	Components	Status	Rationale
UC-1	Set Course	Admin, Course, CourseDAO, FocusCourse, Notifier	Covered	Admin orchestrates course creation, Course and FocusCourse represent data, CourseDAO persists, Notifier sends notifications.
UC-2	Delete Course	Admin, Course, CourseDAO, Notifier	Covered	Admin deletes course via CourseDAO and triggers Notifier.
UC-3	View Employee List	Admin, TrainerController, Employee, EmployeeDAO, Company	Covered	Controllers query EmployeeDAO and Company to build employee lists.
UC-4	Upload Videos and Slides	TrainerController, Material/Slide/Video, MaterialDAO, Notifier	Covered	TrainerController handles upload, Material entities represent content, MaterialDAO persists, Notifier informs companies.
UC-5	Pay Fee	CompanyController, Employee, Subscription, FeeStrategy, SubscriptionDAO, Company	Covered	CompanyController uses SubscriptionDAO and FeeStrategy to compute and update fees for employees of a Company.
UC-6	Register Employee	CompanyController, Employee, Company, Subscription, FeeStrategy, EmployeeDAO, SubscriptionDAO	Covered	CompanyController creates Employee and Subscription, DAOs persist them, FeeStrategy selected based on company employees.

4.3 Use Cases to Tests Mapping

Table 15: Use Cases to Tests Traceability

UC ID	UC Name	Main Flow	Alt Flows	Status & Details
UC-1	Set Course	Tested	Partial	Main flow covered by AdminTest (setCourse, viewCourses). Alternative flow for invalid data is not explicitly mentioned in tests.
UC-2	Delete Course	Tested	Partial	AdminTest covers deleteCourse and verifies removal; no explicit test for error conditions (e.g., non-existing course).
UC-3	View Employee List	Tested	Not Tested	AdminTest and TrainerControllerTest verify retrieval of employees; no alternative flows (e.g., empty list) are described or tested.
UC-4	Upload Videos and Slides	Tested	Partial	TrainerControllerTest covers upload, view and delete of materials; failure scenarios are not clearly described in tests.
UC-5	Pay Fee	Tested	Partial	CompanyControllerTest verifies fee calculation and payment logic; tests for discount edge cases and payment failures are not detailed.
UC-6	Register Employee	Tested	Partial	CompanyControllerTest verifies registration and fee strategy assignment; invalid form input alternative flow is not explicitly tested.

4.4 Complete Traceability Overview

4.4.1 Coverage Summary

Table 16: Traceability Coverage Summary

Artifact Type	Total	Covered	Uncovered	Coverage %
Requirements	18	18 (14 full, 4 partial)	0	100% (77.8% fully)
Use Cases	6	6	0	100% (main flows)
Components	15	15	0	100%
Test Suites	4	4	0	100% (at suite level)

4.4.2 Consolidated Traceability Matrix

Table 17: Consolidated Traceability Matrix (Selected Requirements)

Req ID	Use Cases	Components	Tests	Status
REQ-1	UC-1	Admin, Course, CourseDAO, FocusCourse	AdminTest	Fully Covered
REQ-3	UC-2	Admin, Course, CourseDAO, Notifier	AdminTest	Fully Covered
REQ-6	UC-3	Admin, TrainerController, Employee, EmployeeDAO, Company	AdminTest, TrainerControllerTest	Fully Covered
REQ-9	UC-4	TrainerController, Material, Slide, Video, MaterialDAO, Notifier	TrainerControllerTest	Fully Covered (main flow)
REQ-11	UC-6	CompanyController, Employee, Company, Subscription, FeeStrategy, EmployeeDAO, SubscriptionDAO	CompanyControllerTest	Fully Covered (main flow)
REQ-14	UC-5	CompanyController, Subscription, FeeStrategy, SubscriptionDAO, Employee, Company	CompanyControllerTest	Fully Covered (discount logic tested)
REQ-16	-	ConnectionManager, DAOs, PostgreSQL	TrainerDAOTest, other DAO tests	Fully Covered (architectural)
REQ-17	-	Business Logic, Domain Model, ORM packages	All tests	Fully Covered (structural)

4.4.3 Orphan Artifacts

- **Requirements:** None completely orphan; some (REQ-5, REQ-7, REQ-8, REQ-10, REQ-15) are only partially covered due to missing explicit use cases or tests for alternative/error flows.
- **Use Cases:** None.
- **Tests:** No orphan test suites; all described suites relate to controllers or DAOs that implement use cases.

- **Mockups:** All mockups correspond directly to UC-1–UC-6.

5 Feature-Based Validation

5.1 Overview

The report was evaluated against a conceptual knowledge base of 50 universal software engineering features (problem definition, architecture, testing, documentation, etc.). Based on the available text, approximately **41** of these features are clearly present or partially present, yielding an estimated coverage of **82%**.

Table 18: Feature-Based Validation Summary (Approximate)

Metric	**Value**
Total Knowledge Base Features	50
Features Clearly or Partially Covered	41
Features Not Covered	9
Coverage Percentage	**82.0%**

5.2 Examples of Covered Features

Table 19: Sample Covered Universal Features

ID	**Feature**	**Category**	**Evidence from Report**
F-1	Clear problem context	Problem Definition	"Il presente programma è progettato per gestire un centro di formazione su tematiche legate al settore IT, offrendo funzionalità avanzate per l'organizzazione e la gestione delle iscrizioni..." (Section 1.1).
F-2	Stakeholder and actor identification	Problem Definition	Section 1.1 and 2.1 clearly identify Amministratore, Formatori and Aziende and describe their responsibilities.
F-3	Use case driven design	Architecture	Section 2.1 and 2.2 provide use case diagrams and templates that drive the design of controllers and domain model.
F-4	Layered architecture	Architecture	Section 3.1 describes three main packages (Business Logic, Domain Model, ORM) with distinct responsibilities.
F-5	Use of design patterns	Architecture	Section 3.2.2.2 describes Strategy pattern for fee calculation; Section 3.2.1.4 and 3.2.3.1 describe Singleton pattern for Notifier and ConnectionManager.
F-6	Database schema documentation	Architecture	Section 3.3 presents ER diagram and logical schema of the database.

Continued on next page

Table 19 – continued from previous page

ID	Feature	Category	Evidence from Report
F-7	Systematic unit testing	Testing	Section 4 describes JUnit-based tests for controllers and DAOs, with 51 tests executed successfully.
F-8	Test isolation and cleanup	Testing	Section 4.1 states that each test uses try-catch and cleanup to ensure independence and repeatability.
F-9	Use of version control	Project Management	Section 1.2 mentions GitHub repository for source code management.
F-10	Use of modeling tools	Documentation	Section 1.2 mentions StarUML for UML diagrams and Balsamiq for mockups.
F-11	Use of automated test data generation	Testing	Section 1.2 mentions integration with ChatGPT to generate automated test data.
F-12	Technical documentation using LaTeX	Documentation	Section 1.2 mentions Overleaf and LaTeX for the technical report.

5.3 Examples of Uncovered or Weakly Covered Features

Table 20: Sample Uncovered or Weakly Covered Features

ID	Feature	Category	Description
UF-1	Explicit performance requirements	Performance	No quantitative performance targets (e.g., response time, throughput) are specified.
UF-2	Security requirements and threat analysis	Security	No explicit security requirements (authentication, authorization, data protection) or threat analysis are documented.
UF-3	Error handling and recovery strategy	Architecture	Error handling is only briefly mentioned in some use cases; no global strategy is described.
UF-4	Deployment architecture	Architecture	No deployment topology (e.g., server layout, environment) is described.
UF-5	Monitoring and logging strategy	Maintainability	No description of logging, monitoring or observability is provided.
UF-6	Scalability considerations	Performance	No discussion of how the system scales with number of users or data volume.
UF-7	Usability metrics	Documentation	Mockups are provided, but no usability goals or metrics are defined.

Continued on next page

Table 20 – continued from previous page

ID	Feature	Category	Description
UF-8	Configuration management	Project Management	No description of configuration or environment management.
UF-9	Continuous integration / delivery	Project Management	No mention of CI/CD pipelines or automated build processes.

5.4 Checklist Compliance Example: Layered Architecture

Feature: Layered architecture pattern
Description: System uses a layered architecture with clear separation of concerns.

Table 21: Checklist Compliance for "Layered architecture pattern"

ID	Checklist Item	Status	Explanation
1.1	Architecture pattern is identified and justified	True	Section 3.1 explicitly describes three main packages with distinct responsibilities, justifying separation of concerns.
1.2	Layer responsibilities are clearly defined	True	Business Logic, Domain Model and ORM responsibilities are described in detail.
1.3	Communication between layers is specified	Partial	Interactions between controllers, domain entities and DAOs are described, but interface contracts and data formats are not fully specified.
1.4	Dependencies flow in one direction	True	Domain Model is independent of ORM; DAOs depend on ConnectionManager and domain entities; controllers depend on domain and DAOs. No reverse dependencies are indicated.

6 Detailed Analysis

6.1 Requirements Quality Assessment

6.1.1 Issues and Recommendations

Requirements needing more detail:

- **REQ-4 (notifications on course changes):** Does not specify timing (immediate vs. batch), failure handling or email content. **Recommendation:** Define when notifications are sent and what happens if email delivery fails.

- **REQ-5 (workshift management):** Groups several operations; does not specify constraints (e.g., overlapping shifts). **Recommendation:** Split into separate requirements and define business rules for scheduling.
- **REQ-6 (view employee data):** Does not specify access control or privacy constraints. **Recommendation:** Clarify which actor can see which fields and under what conditions.
- **REQ-7 (payment reminders):** Does not specify reminder frequency or channels. **Recommendation:** Define schedule (e.g., weekly) and escalation rules.
- **REQ-10 (material notifications):** Only upload success is mentioned. **Recommendation:** Clarify whether updates and deletions also trigger notifications.
- **REQ-13 (fee payment):** Payment method and error handling are not described. **Recommendation:** Specify payment mechanisms (e.g., bank transfer, card) and what happens on failure.

Vague or high-level requirements:

- **REQ-15 (centralized notifications):** Good design principle but not measurable. **Recommendation:** Add acceptance criteria (e.g., "All emails must be sent via Notifier; no direct SMTP calls elsewhere").
- **REQ-18 (overall goal):** High-level and non-testable. **Recommendation:** Keep as background but ensure it is refined into concrete functional and non-functional requirements (which is partially done via use cases).

6.2 Use Case Completeness Analysis

- Main flows for the six key use cases are clearly described and aligned with actors and architecture.
- Alternative flows are present for UC-1, UC-4 and UC-6 (input validation and upload failure), but missing for UC-2, UC-3 and UC-5.
- Error scenarios (e.g., database failures, network issues) are not modeled as alternative flows.

Recommendation: Extend use case templates to include:

- Error handling for course deletion when the course no longer exists (UC-2).
- Behavior when no employees or courses are available (UC-3, UC-5).
- Payment failure scenarios (UC-5).

6.3 Architecture Quality Evaluation

Table 22: Architecture Quality Metrics

Principle	Status	Notes
Separation of Concerns	Good	Clear package boundaries and DAO separation.
Loose Coupling	Good	Domain model is independent of persistence; DAOs encapsula
High Cohesion	Moderate	Controllers aggregate many operations; could be decomposed in
Single Responsibility	Moderate	Admin and CompanyController handle multiple concerns (courses, emp

Recommendations:

- Introduce service classes (e.g., CourseService, EmployeeService, BillingService) to reduce controller responsibilities.
- Document error handling and transaction boundaries in the ORM layer.
- Add a brief description of the presentation technology (e.g., JavaFX, web framework) to complete the architecture picture.

6.4 Test Coverage Analysis

- Controllers and DAOs are well-covered by unit tests, with 51 tests executed successfully.
- Tests follow a logical order for DAOs (insert, retrieve, update, delete) and include cleanup.
- There is no explicit mention of integration or system-level tests that exercise full end-to-end flows including UI.

Recommendations:

- Add a small set of integration tests that simulate complete user journeys (e.g., register employee then pay fee).
- Document representative test method names and their mapping to use case steps for improved traceability.
- Consider adding negative tests for error scenarios (invalid input, database errors).

7 Recommendations

7.1 Priority 1: Critical Items

1. **Clarify Non-Functional Requirements**
   - Add explicit performance, security and usability requirements.
   - **Action:** Create a dedicated "Requisiti Non-Funzionali" section with measurable criteria.
2. **Extend Use Cases with Error and Alternative Flows**
   - UC-2, UC-3 and UC-5 lack alternative flows.
   - **Action:** Add error scenarios (e.g., payment failure, missing data) and document system behavior.
3. **Improve Test Traceability**
   - Current tests are described at class level.
   - **Action:** Document a table mapping key test methods to specific use case steps and alternative flows.

7.2 Priority 2: Important Improvements

1. **Refine Controller Responsibilities**
   - Controllers currently handle multiple concerns.
   - **Action:** Introduce service classes to encapsulate course management, employee management and billing logic.
2. **Document Error Handling and Transactions**
   - ORM layer behavior on failures is not described.
   - **Action:** Add a subsection describing transaction management, rollback behavior and exception handling strategy.
3. **Add Deployment and Environment Description**
   - No deployment architecture is provided.
   - **Action:** Include a simple deployment diagram and description of runtime environment (e.g., single server, local DB).

7.3 Priority 3: Nice-to-Have Enhancements

1. Add basic performance tests or benchmarks for critical DAO operations.
2. Define logging and monitoring guidelines (e.g., what to log in controllers and DAOs).
3. Provide a short user manual or quick-start guide based on the mockups and use cases.

7.4 Summary Checklist

Table 23: Action Item Summary

Priority	**Items**	**Indicative Effort**
Critical (P1)	3	2–3 days
Important (P2)	3	3–5 days
Nice-to-Have (P3)	3	3–5 days
Total	**9**	**8–13 days**