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	16
Total Use Cases Extracted	4 (4 explicit, 0 implicit)
Total Architecture Components	8
Total Tests Extracted	34
Requirements Coverage	100% (16/16 mapped to at least one use case)
Use Case Coverage by Tests	100% (4/4 have main flow tests; alt flows partially covered)
Architecture Mapping Coverage	100% (4/4 use cases mapped to components)
Feature-Based Coverage (vs. generic SE best practices)	78.0% (approximate, qualitative)
Overall Status	**PASSED WITH MINOR GAPS**

1.2 Key Findings

- **Strengths:**
  - Clear, explicit use case templates for the main user and admin flows (purchase, library, catalog, subscriptions).
  - Well-structured layered architecture (CLI, Business Logic, ORM, JDBC, PostgreSQL) with good separation of concerns.
  - Extensive JUnit 5 test suite covering business logic, domain model pricing strategy, and ORM DAOs, including many failure scenarios.
  - Consistent use of design patterns (Strategy, Observer, Singleton, DAO) and clear explanation of their roles.
- **Areas for Improvement:**
  - Functional and non-functional requirements are implicit in use cases and code; there is no dedicated requirements section and some qualities (performance, security, usability) are not quantified.
  - Some alternative flows in use cases are only partially covered by tests (e.g., some admin validation errors, some library edge cases).
  - No explicit discussion of security, performance, scalability, or deployment architecture beyond basic JDBC/PostgreSQL usage.
  - Error handling and logging strategies are implemented in code but not documented as architectural policies.

2 Content Extraction Results

2.1 Table of Contents and Sections

2.1.1 Extracted Table of Contents

Table 2: Extracted Table of Contents (from PDF)

Section	**Start Page**	**End Page**
1 Introduzione	4	5
1.1 Statement del progetto	4	4
1.2 Architettura e strumenti	4	5
1.3 Relazioni tra i pacchetti	4	5
2 Documentazione	6	12
2.1 Use Case Diagram	6	6
2.2 Use Case Template	7	9
2.3 Mock-Ups	9	11
2.4 Class Diagram	12	12
2.5 ER Diagram	12	12
3 Implementazione	13	25
3.1 Domain Model	13	15
3.1.1 Utente	13	13
3.1.2 Videogioco	13	13
3.1.3 Carrello	13	13
3.1.4 Libreria	13	14
3.1.5 Abbonamento	14	14
3.1.6 StatisticheVideogioco	14	14
3.1.7 Achievement	14	14
3.1.8 Admin	14	14
3.2 Business Logic	14	19
3.2.1 UtenteController	14	15
3.2.2 CarrelloController	15	16
3.2.3 LibreriaController	16	17
3.2.4 CatalogoController	17	18
3.2.5 StatisticheController	18	18
3.2.6 AchievementController	18	19
3.2.7 AdminController	19	19
3.3 ORM (Object-Relational Mapping)	19	24
3.3.1 DatabaseConnection	19	19
3.3.2 UtenteDAO	19	20
3.3.3 VideogiocoDAO	20	21
3.3.4 CarrelloDAO	21	22
3.3.5 LibreriaDAO	21	23
3.3.6 AbbonamentoDAO	22	23
3.3.7 StatisticheVideogiocoDAO	23	23
3.3.8 AchievementDAO	23	24
3.3.9 AdminDAO	24	24
3.4 Database	24	24
4 Test	25	40
4.1 Business Logic Test	25	33
4.1.1 UtenteControllerTest	25	26
4.1.2 CatalogoControllerTest	26	27
4.1.3 CarrelloControllerTest	27	29
4.1.4 LibreriaControllerTest	29	31

Continued on next page

Table 2 – continued

Section	Start Page	End Page
4.1.5 StatisticheControllerTest	31	32
4.1.6 AchievementControllerTest	32	33
4.1.7 AdminControllerTest	33	33
4.2 Domain Model Test	34	35
4.2.1 CarrelloTest	35	36
4.3 Test del Pacchetto ORM	36	40
4.3.1 VideogiocoDAOTest	36	37
4.3.2 CarrelloDAOTest	37	39
4.3.3 LibreriaDAOTest	39	40

2.2 Section Classification

Table 3: Section Classification by Content Type

Section	Labels
1.1 Statement del progetto	Requirements
1.2 Architettura e strumenti	Architecture
1.3 Relazioni tra i pacchetti	Architecture
2.1 Use Case Diagram	Use Cases
2.2 Use Case Template	Use Cases, Requirements, Test (references)
2.3 Mock-Ups	Mockups
2.4 Class Diagram	Architecture
2.5 ER Diagram	Architecture
3.1 Domain Model	Architecture
3.2 Business Logic (and subsections)	Architecture, Use Cases (implementation of UC templates)
3.3 ORM (and subsections)	Architecture
3.4 Database	Architecture, Test
4.1 Business Logic Test (and subsections)	Test
4.2 Domain Model Test	Test
4.3 Test del Pacchetto ORM (and subsections)	Test

2.3 Requirements Extraction

The report does not contain a dedicated requirements chapter; requirements are implicit in the project statement and use case templates. The following atomic requirements were extracted.

2.3.1 Requirements Quality Distribution

Table 4: Requirements Quality Assessment

Quality Level	**Count**	**Percentage**
Well-defined	9	56.3%
Needs Detail	5	31.3%
Vague/Unquantified	2	12.5%
Total	**16**	**100%**

2.3.2 Requirements by Type

Table 5: Requirements Classification

Type	**Count**	**Percentage**
Functional	13	81.3%
Non-Functional	1	6.3%
Constraint	1	6.3%
Goal/Background	1	6.3%
Total	**16**	**100%**

2.3.3 Detailed Requirements List

Table 6: Extracted Requirements

ID	**Type**	**Description**	**Quality**
REQ-1	goal/background	The system must provide a platform for managing a videogame sales system, allowing users to explore a catalog, purchase games, and manage a personal library.	Needs Detail
REQ-2	functional	Users must be able to explore a catalog of videogames and perform searches based on criteria such as genre, platform, and release date.	Needs Detail
REQ-3	functional	Users must be able to add selected videogames from the catalog to a virtual cart for purchase.	Well-defined
REQ-4	functional	After purchase, games must be added to the user's personal library, where the user can install, uninstall, or start games to play.	Well-defined
REQ-5	functional	During game sessions, the system must record gameplay statistics such as total play time.	Needs Detail
REQ-6	functional	The system must support unlocking achievements associated with videogames based on gameplay statistics.	Needs Detail

Continued on next page

Table 6 – continued from previous page

ID	Type	Description	Quality
REQ-7	functional	The system must provide two types of monthly subscriptions (Gold and Silver) that offer discounts on games and access to a selection of free games.	Needs Detail
REQ-8	functional	Users must have a monetary fund that they can recharge and that is used to pay for subscriptions and games in the cart.	Well-defined
REQ-9	functional	The system must provide an administrator role with functions to add, remove, and modify games in the catalog.	Well-defined
REQ-10	functional	The administrator must be able to select free titles for subscribers and add, remove, and modify achievements associated with games.	Well-defined
REQ-11	constraint	The user interface must be implemented as a Command Line Interface (CLI).	Well-defined
REQ-12	non-functional	The architecture must be modular and layered, with distinct packages for domain model, business logic, ORM, and database access to ensure clarity, maintainability, and scalability.	Vague/Unquantified
REQ-13	functional	Only authenticated users must be allowed to access user functionalities such as catalog browsing, cart management, library management, and subscription management.	Well-defined
REQ-14	functional	Only authenticated administrators must be allowed to perform administrative operations on the catalog and achievements.	Well-defined
REQ-15	functional	The system must prevent users from purchasing games that are already present in their library.	Well-defined
REQ-16	functional	The system must prevent users from activating a subscription or completing a purchase if their fund is insufficient, and must notify them accordingly.	Well-defined

Notes on Requirement Quality

- Several requirements (e.g., REQ-2, REQ-5, REQ-6, REQ-7, REQ-12) would benefit from quantitative criteria (e.g., maximum response time, specific discount percentages, limits on statistics retention).
- Non-functional aspects such as performance, security, and usability are largely implicit and not specified with measurable metrics.

2.4 Use Case Extraction

2.4.1 Use Case Summary

Table 7: Extracted Use Cases

ID	Name	Type	Actors
UC-001	Purchase a videogame	Explicit	Utente (User)
UC-002	Manage personal library	Explicit	Utente (User)
UC-003	Manage catalog as administrator	Explicit	Admin
UC-004	Manage subscriptions	Explicit	Utente (User)

2.4.2 Detailed Use Case Descriptions

UC-001: Purchase a videogame

- **Actors:** User
- **Type:** Explicit
- **Preconditions:** The user is authenticated and is in the main menu.
- **Main Flow:**
  1. The user selects the option to access the catalog.
  2. The user chooses between viewing all games or performing a search.
  3. The user views the complete list of purchasable games or the filtered list based on search criteria.
  4. The user selects a videogame to purchase and adds it to the cart.
  5. The user can view the cart and the total cost.
  6. The user executes the purchase of the games in the cart.
- **Alternative Flows:**
  - 4a. If the user selects a game already purchased, the system notifies the user and they retry.
  - 6a. If the user accesses an empty cart, the system shows a message and returns to the catalog.
  - 6b. If the user's fund is insufficient, the system notifies the user and the purchase cannot be completed.
- **Postconditions:** The cart is emptied, the games are moved to the user's personal library and saved in the system, and the fund is updated.

UC-002: Manage personal library

- **Actors:** User
- **Type:** Explicit
- **Preconditions:** The user has logged in and owns purchased games.
- **Main Flow:**
  1. The user selects the option to access their game library.

2. The system shows the games purchased by the user with description and installation status.
3. The user selects to install or uninstall a game.
4. The system updates the installation status or the play time associated with the selected videogame.

- **Alternative Flows:**
  - 3a. If the library is empty, the user receives a message and cannot perform actions.
  - 3b. The user selects the option to start a videogame and play:
    - 3b.1 The user selects how long they want to play.
    - 3b.2 The user simulates playing and the play time is updated.
  - 3c. The user selects the option to view statistics and achievements for a videogame:
    - 3c.1 The system shows the statistics associated with the game.
    - 3c.2 If there are no achievements associated with the game, the system sends a message.
  - 3d. If the user selects to install an already installed game or uninstall a non-installed game, the system prevents it and sends a message.
- **Postconditions:** The system updates the installation status or play time for the selected videogame and saves the changes in the database.

UC-003: Manage catalog as administrator

- **Actors:** Admin
- **Type:** Explicit
- **Preconditions:** The administrator has logged in with valid credentials.
- **Main Flow:**
  1. The admin accesses the admin menu.
  2. The admin selects an action: add, modify, delete videogame or achievement, or set/remove free status.
  3. The system executes the operation and shows a confirmation.
- **Alternative Flows:**
  - 1a. If credentials are not valid, an error message is shown.
  - 2a. If the data entered by the admin during creation or modification are not valid, an error message is shown.
- **Postconditions:** The catalog is updated in the database and changes to videogames and achievements are saved.

UC-004: Manage subscriptions

- **Actors:** User
- **Type:** Explicit
- **Preconditions:** The user has logged in with valid credentials.
- **Main Flow:**
  1. The user selects the option to manage subscriptions from the user menu.
  2. The system shows the current subscription status (none, Gold, Silver) and asks which subscription to activate.
  3. The user selects an option.
  4. The system verifies that the user's fund is sufficient for the subscription cost.
  5. The system charges the cost and activates the subscription.
- **Alternative Flows:**
  - 3a. If a subscription is already active, the user is notified with a message.
  - 4a. If the fund is insufficient, the system shows an error and proposes to recharge the fund.
- **Postconditions:** The subscription status is updated in the database and benefits (discounts and free games) are applied.

2.5 Architecture Extraction

2.5.1 Architectural Pattern

The report describes a clear **Layered Architecture** with modular packages:

- CLI (user interface)
- Business Logic (controllers and application services)
- Domain Model (entities)
- ORM / DAO layer
- JDBC
- RDBMS (PostgreSQL)

2.5.2 Architecture Components

Table 8: Architecture Components Analysis

Component	Responsibility	Design Notes
CLI	Command-line user interface for menus, catalog browsing, cart, library, admin menu.	Clear separation from business logic; acts as presentation layer.

Continued on next page

Table 8 – continued from previous page

Component	Responsibility	Design Notes
Business Logic Controllers (UtenteController, CarrelloController, LibreriaController, CatalogoController, StatisticheController, AchievementController, AdminController)	Implement operations defined in use cases: registration, authentication, cart management, purchases, library operations, statistics, achievements, admin catalog management.	Responsibilities are well described; some controllers (CarrelloController, CatalogoController) are central but still focused. Good use of Strategy and Observer patterns.
Domain Model Entities (Utente, Videogioco, Carrello, Libreria, Abbonamento, StatisticheVideogioco, Achievement, Admin)	Represent core business entities and their attributes and simple behaviors (getters/setters, some logic like price calculation via strategy).	Entities are mostly data containers; Carrello integrates Strategy pattern via PrezzoStrategy. Cohesion is good.
PrezzoStrategy and concrete strategies	Calculate videogame prices depending on subscription type (Gold, Silver, NonAbbonato).	Classic Strategy pattern; improves flexibility for pricing rules.
Observer (CarrelloController as Subject, LibreriaController as Observer)	Notify library when a purchase is completed so that games are added and messages are shown.	Good use of Observer to decouple purchase from library update; security checks in observer.
ORM / DAO Layer (UtenteDAO, VideogiocoDAO, CarrelloDAO, LibreriaDAO, AbbonamentoDAO, StatisticheVideogiocoDAO, AchievementDAO, AdminDAO)	Provide CRUD operations for each entity, mapping domain objects to PostgreSQL tables.	Clear DAO responsibilities; SQL is parameterized; methods like findByCriteria support flexible queries.
DatabaseConnection (Singleton)	Provide a single access point to PostgreSQL connections using JDBC.	Singleton pattern; simple but effective for this context.
PostgreSQL Database	Store all persistent data: users, games, carts, libraries, subscriptions, statistics, achievements, admins.	Schema documented via ER diagram; test scripts ensure consistent state.

2.5.3 Architecture Analysis

Summary: The architecture follows a clear layered pattern with good separation of concerns. Presentation (CLI), business logic, domain model, and persistence are separated. Design patterns (Strategy, Observer, Singleton, DAO) are used appropriately.

Strengths:

- Clear layering and modular packages.
- DAOs encapsulate SQL and database access, keeping business logic independent of persistence details.
- Strategy pattern cleanly encapsulates pricing rules based on subscription.
- Observer pattern decouples purchase completion from library updates.

Observations and Minor Gaps:

- No explicit discussion of transaction management (e.g., ensuring atomicity across multiple DAO calls during purchase).
- Error handling is implemented via exceptions but there is no documented global error-handling policy.
- Security is limited to simple credential checks; there is no mention of password hashing or protection against SQL injection beyond prepared statements.
- Deployment and scalability aspects (e.g., multi-user concurrency, connection pooling) are not discussed.

2.6 Test Extraction

2.6.1 Test Type Distribution

All tests are written with JUnit 5 and exercise business logic, domain model, and DAOs. They are effectively **integration tests** (because they hit a real PostgreSQL database) plus some unit-like tests for Carrello.

Table 9: Test Distribution by Type (Qualitative)

Test Type	Count	Percentage
Integration (Business Logic + DB)	24	70.6%
Unit-like (Domain Model Carrello)	7	20.6%
Integration (DAO-level)	3	8.8%
Total	**34**	**100%**

2.6.2 Detailed Test List

Table 10: Extracted Tests (Representative)

ID	Type	Artifact	Coverage Hint
TEST-1	Integration	UtenteControllerTest.testRegistrazioneUtente_Successo	User registration success scenario
TEST-2	Integration	UtenteControllerTest.testRegistrazioneUtente_UsernameEsistente	User registration with existing username error handling
TEST-3	Integration	UtenteControllerTest.testAutenticazioneUtente_CredenzialiValide	User login with valid credentials
TEST-4	Integration	UtenteControllerTest.testRicaricaFondi_Successo	User fund recharge success
TEST-5	Integration	UtenteControllerTest.testAttivaAbbonamento_Successo	Subscription activation with sufficient funds
TEST-6	Integration	UtenteControllerTest.testAttivaAbbonamento_FallimentoFondiInsufficienti	Subscription activation failure due to insufficient funds
TEST-7	Integration	CatalogoControllerTest.testAggiungiVideogioco_Successo	Admin adds new videogame to catalog
TEST-8	Integration	CatalogoControllerTest.testAggiungiVideogioco_FallimentoVideogiocoEsistente	Admin attempts to add duplicate videogame
TEST-9	Integration	CatalogoControllerTest.testCercaVideogiochi_Successo	Search videogames by genre with results
TEST-10	Integration	CatalogoControllerTest.testCercaVideogiochi_FallimentoNessunRisultato	Search videogames with no results
TEST-11	Integration	CarrelloControllerTest.testAggiungiAlCarrello_Successo	Add videogame to cart
TEST-12	Integration	CarrelloControllerTest.testRimuoviDalCarrello_Successo	Remove videogame from cart
TEST-13	Integration	CarrelloControllerTest.testEseguiAcquisto_Successo	Execute purchase with sufficient funds
TEST-14	Integration	CarrelloControllerTest.testEseguiAcquisto_FallimentoFondiInsufficiente	Purchase failure due to insufficient funds
TEST-15	Integration	CarrelloControllerTest.testEseguiAcquisto_FallimentoVideogiocoGiaInLibreria	Purchase failure when game already in library
TEST-16	Integration	CarrelloControllerTest.testAggiungiRimuoviObserver_Successo	Add and remove library observer from cart
TEST-17	Integration	CarrelloControllerTest.testNotificaAcquistoCompletato_Successo	Notify observers on completed purchase
TEST-18	Integration	LibreriaControllerTest.testCaricaLibreria_Successo	Load user library
TEST-19	Integration	LibreriaControllerTest.testInstallaVideogioco_Successo	Install purchased videogame

Continued on next page

Table 10 – continued from previous page

ID	Type	Artifact	Coverage Hint
TEST-20	Integration	LibreriaControllerTest.testInstallaVideogioco_Fallimento_VideogiocoGiaInstalla	Install already installed videogame error
TEST-21	Integration	LibreriaControllerTest.testDisinstallaVideogioco_Successo	Uninstall installed videogame
TEST-22	Integration	LibreriaControllerTest.testDisinstallaVideogioco_Fallimento_VideogiocoNonInst	Uninstall non-installed videogame error
TEST-23	Integration	LibreriaControllerTest.testOnAcquistoCompletato_Successo	Library updated after purchase via observer
TEST-24	Integration	StatisticheControllerTest.testGetStatistiche_Successo	Retrieve existing gameplay statistics
TEST-25	Integration	StatisticheControllerTest.testGetStatistiche_Fallimento_NessunaStatistica	No statistics for user/game combination
TEST-26	Integration	StatisticheControllerTest.testGetUtenteAchievements_Successo	Retrieve user achievements for a game
TEST-27	Integration	StatisticheControllerTest.testGetUtenteAchievements_Fallimento_NessunAchiev	No achievements for user/game combination
TEST-28	Integration	AchievementControllerTest.testAggiungiAchievement_Successo	Admin adds new achievement
TEST-29	Integration	AchievementControllerTest.testAggiungiAchievement_Fallimento_AdminNonAu	Add achievement with unauthenticated admin
TEST-30	Integration	AchievementControllerTest.testGetAchievementsByVideogioco_Successo	Retrieve achievements by videogame
TEST-31	Integration	AchievementControllerTest.testGetAchievementsByVideogioco_Fallimento_Ness	No achievements for videogame
TEST-32	Integration	AdminControllerTest.testRegistraAdmin_Fallimento_UsernameEsistente	Register admin with existing username
TEST-33	Integration	AdminControllerTest.testAutenticaAdmin_Successo	Admin authentication success
TEST-34	Integration	AdminControllerTest.testAutenticaAdmin_Fallimento_UsernameNonEsistente	Admin authentication with non-existing username
TEST-35	Unit-like	CarrelloTest.testAddVideogioco_Successo_NuovoVideogioco	Add new videogame to cart entity
TEST-36	Unit-like	CarrelloTest.testAddVideogioco_Fallimento_VideogiocoDuplicato	Prevent duplicate videogame in cart entity
TEST-37	Unit-like	CarrelloTest.testRemoveVideogioco_Successo	Remove videogame from cart entity
TEST-38	Unit-like	CarrelloTest.testSetPrezzoStrategy_Successo	Set Gold pricing strategy and compute total

Continued on next page

Table 10 – continued from previous page

ID	Type	Artifact	Coverage Hint
TEST-39	Unit-like	CarrelloTest.testGetTotale_NonAbbonato_GiocoAPagamento	Compute total for non-subscriber with paid game
TEST-40	Unit-like	CarrelloTest.testGetTotale_GoldAbbonato_GiocoAPagamento	Compute total for Gold subscriber with paid game
TEST-41	Unit-like	CarrelloTest.testGetTotale_GoldAbbonato_GiocoGratuito	Compute total for Gold subscriber with free game
TEST-42	Integration	VideogiogoDAOTest.testSave_Successo	Save new videogame in DB
TEST-43	Integration	VideogiogoDAOTest.testUpdate_Successo	Update videogame price in DB
TEST-44	Integration	VideogiogoDAOTest.testFindById_Successo	Find videogame by ID
TEST-45	Integration	VideogiogoDAOTest.testFindById_Fallimento_IdNonEsistente	Find videogame by non-existing ID
TEST-46	Integration	VideogiogoDAOTest.testFindByCriteria_Successo_TuttiCriteri	Find videogames by full criteria
TEST-47	Integration	VideogiogoDAOTest.testFindByCriteria_Fallimento_NessunRisultato	Criteria with no results
TEST-48	Integration	VideogiogoDAOTest.testDelete_Successo	Delete videogame from DB
TEST-49	Integration	CarrelloDAOTest.testSave_Successo_CarrelloEsistente	Save cart with new videogame
TEST-50	Integration	CarrelloDAOTest.testUpdate_Successo	Update cart contents
TEST-51	Integration	CarrelloDAOTest.testFindByUtenteId_Successo	Find cart by user ID
TEST-52	Integration	CarrelloDAOTest.testFindByUtenteId_Fallimento_UtenteNonEsistente	No cart for non-existing user
TEST-53	Integration	CarrelloDAOTest.testDelete_Successo	Delete cart from DB
TEST-54	Integration	LibreriaDAOTest.testSave_Successo_NuovaLibreria	Save new library with videogame and installation state
TEST-55	Integration	LibreriaDAOTest.testFindByUtenteId_Successo	Find library by user ID
TEST-56	Integration	LibreriaDAOTest.testFindByUtenteId_Fallimento_UtenteNonEsistente	No library for non-existing user
TEST-57	Integration	LibreriaDAOTest.testUpdate_Successo	Update library installation state

3 Traceability Matrix

3.1 Requirements to Use Cases Mapping

Table 11: Requirements to Use Cases Traceability

Req ID	Use Cases	Status	Rationale
REQ-1	UC-001, UC-002, UC-004	Covered	The overall platform goal is realized by purchase, library management, and subscription management use cases.
REQ-2	UC-001	Covered	UC-001 describes catalog exploration and search before adding to cart.
REQ-3	UC-001	Covered	UC-001 explicitly includes adding games to a virtual cart.
REQ-4	UC-001, UC-002	Covered	UC-001 moves purchased games to the library; UC-002 manages install/uninstall/start.
REQ-5	UC-002	Covered	UC-002 alternative flow 3b describes updating play time during simulated play.
REQ-6	UC-002	Covered	UC-002 alternative flow 3c covers viewing statistics and achievements.
REQ-7	UC-004	Covered	UC-004 describes choosing and activating Gold or Silver subscriptions.
REQ-8	UC-001, UC-004	Covered	UC-001 and UC-004 both rely on the user fund to complete purchases and subscriptions.
REQ-9	UC-003	Covered	UC-003 describes admin operations to add, modify, and delete games.
REQ-10	UC-003	Covered	UC-003 includes managing achievements and free status for subscribers.
REQ-11	UC-001, UC-002, UC-003, UC-004	Covered	All use cases are described as being executed via CLI menus.
REQ-12	UC-001, UC-002, UC-003, UC-004	Covered	All use cases are implemented using the layered architecture described in the implementation sections.
REQ-13	UC-001, UC-002, UC-004	Covered	Preconditions of these use cases require the user to be authenticated.
REQ-14	UC-003	Covered	UC-003 precondition requires admin login with valid credentials.
REQ-15	UC-001	Covered	UC-001 alternative flow 4a prevents adding already purchased games; CarrelloController enforces this.

Continued on next page

Table 11 – continued from previous page

Req ID	Use Cases	Status	Rationale
REQ-16	UC-001, UC-004	Covered	UC-001 and UC-004 alternative flows describe insufficient fund scenarios blocking operations.

3.2 Use Cases to Architecture Mapping

Table 12: Use Cases to Architecture Traceability

UC ID	UC Name	Components	Status	Rationale
UC-001	Purchase a videogame	CLI, CatalogoController, CarrelloController, UtenteController, LibreriaController, VideogiocoDAO, CarrelloDAO, LibreriaDAO, UtenteDAO	Covered	CLI drives the flow; controllers orchestrate catalog, cart, user fund, and library; DAOs persist changes.
UC-002	Manage personal library	CLI, LibreriaController, StatisticheController, LibreriaDAO, StatisticheVideogiocoDAO, AchievementDAO	Covered	LibreriaController handles install/uninstall/start; StatisticheController updates time and achievements; DAOs persist state.
UC-003	Manage catalog as administrator	CLI, AdminController, CatalogoController, AchievementController, VideogiocoDAO, AchievementDAO, AdminDAO	Covered	AdminController authenticates admin; CatalogoController and AchievementController manage catalog and achievements; DAOs handle persistence.
UC-004	Manage subscriptions	CLI, UtenteController, AbbonamentoDAO, UtenteDAO, CarrelloController (for pricing effects)	Covered	UtenteController activates subscriptions and updates fund; AbbonamentoDAO and UtenteDAO persist; CarrelloController uses subscription for pricing.

3.3 Use Cases to Tests Mapping

Table 13: Use Cases to Tests Traceability

UC ID	UC Name	Main Flow	Alt Flows	Status & Details
UC-001	Purchase a videogame	Tested	Partial	Main flow (add to cart, execute purchase, update fund and library) is tested by TEST-11, TEST-12, TEST-13, and DAO tests. Alt flow 6b (insufficient funds) is tested by TEST-14. Alt flow 4a (already purchased game) is tested by TEST-15. Alt flow 6a (empty cart) is not explicitly covered by a dedicated test.
UC-002	Manage personal library	Tested	Partial	Main flow (load library, install/uninstall) is tested by TEST-18, TEST-19, TEST-21. Alt flows for already installed/non-installed games are tested by TEST-20 and TEST-22. Alt flows 3b (start game and update time) and 3c (view statistics and achievements, no achievements) are partially covered via StatisticheControllerTest (TEST-24–TEST-27) but not fully integrated with LibreriaController in tests. Alt flow 3a (empty library) is not explicitly tested.
UC-003	Manage catalog as administrator	Tested	Partial	Main flow (admin adds games, searches catalog) is tested by TEST-7, TEST-9, and DAO tests. Alt flow 1a (invalid credentials) is covered by AdminControllerTest (TEST-34). Alt flow 2a (invalid data) is partially covered by TEST-8 (duplicate videogame) and AchievementController tests, but not all validation errors are explicitly tested.

Continued on next page

Table 13 – continued from previous page

UC ID	UC Name	Main Flow	Alt Flows	Status & Details
UC-004	Manage subscriptions	Tested	Partial	Main flow (activate subscription with sufficient funds) is tested by TEST-5. Alt flow 4a (insufficient funds) is tested by TEST-6. Alt flow 3a (subscription already active) is not explicitly covered by a dedicated test.

3.4 Complete Traceability Overview

3.4.1 Coverage Summary

Table 14: Traceability Coverage Summary

Artifact Type	Total	Covered	Uncovered	Coverage %
Requirements	16	16	0	100%
Use Cases	4	4	0	100%
Components	8	8	0	100%
Use Case Main Flows	4	4	0	100%
Use Case Alt Flows	11	7 (fully) + 4 (not explicit)	–	~64% fully tested

3.4.2 Orphan Artifacts

All extracted requirements are mapped to at least one use case and to implementation components. All use cases have at least one associated test for the main flow. No clear orphan requirements, use cases, or tests were identified based on the provided text.

4 Feature-Based Validation

4.1 Overview

This section evaluates the report qualitatively against universal software engineering best practices. Because there is no external knowledge base provided, coverage percentages are approximate and based on typical expectations for a course project.

Table 15: Feature-Based Validation Summary (Qualitative)

Metric	Value
Total Reference Best-Practice Features (assumed)	50
Features Clearly Covered in Report	~39
Features Not Evident in Report	~11
Coverage Percentage	**78.0% (approx.)**

4.2 Selected Covered Features

Table 16: Examples of Covered Universal Features

ID	Feature	Category	Evidence from Report
F-1	Clear problem statement	Problem Definition	"L'obiettivo del progetto e lo sviluppo di un'applicazione per la gestione di un sistema di vendita di videogiochi." (Section 1.1)
F-2	Use case driven specification	Requirements	Section 2.2 provides detailed use case templates (UC-001 to UC-004) with preconditions, steps, alternative flows, and postconditions.
F-3	Layered architecture	Architecture	Section 1.2 and 1.3 describe a modular, layered architecture with Domain Model, Business Logic, ORM, JDBC, and PostgreSQL.
F-4	Use of design patterns	Architecture	Domain Model and Business Logic sections describe Strategy (PrezzoStrategy), Observer (CarrelloController/LibreriaController), Singleton (DatabaseConnection), and DAO patterns.
F-5	Database schema documentation	Architecture	Section 2.5 presents an ER diagram representing the structure of the tables created by the DBMS.
F-6	Automated tests with realistic data	Testing	Section 4 describes JUnit 5 tests using SQL scripts (reset.sql, ProgettoSWE.sql, InserimentoProgettoSWE.sql) to initialize the database.
F-7	Negative and edge case testing	Testing	Many tests explicitly cover failure scenarios (e.g., insufficient funds, duplicate usernames, non-existing IDs, invalid admin).
F-8	Separation of concerns	Architecture	Business Logic uses DAOs for persistence and does not interact directly with JDBC; CLI only invokes controller methods.
F-9	Reproducible test environment	Testing	Section 3.4 explains that SQL scripts are executed beforeEach and afterEach to ensure a consistent DB state for tests.
F-10	Documentation of main entities	Documentation	Section 3.1 describes each domain entity (Utente, Videogioco, Carrello, Libreria, Abbonamento, StatisticheVideogioco, Achievement, Admin) with roles and attributes.

4.3 Examples of Uncovered or Weakly Covered Features

Table 17: Examples of Uncovered or Weakly Covered Features

ID	Feature	Category	Description
UF-1	Performance requirements	Performance	No explicit performance targets (e.g., response time, throughput) are specified.
UF-2	Security requirements and threat analysis	Security	Security is limited to simple credential checks; there is no discussion of password storage, SQL injection, or threat modeling.
UF-3	Usability and accessibility	Documentation	The CLI is described, but there are no usability or accessibility criteria or evaluations.
UF-4	Deployment architecture	Architecture	No description of deployment topology (e.g., single machine, client-server, containerization) is provided.
UF-5	Logging and monitoring strategy	Maintainability	Logging is used in some methods (System.out.println) but there is no documented logging or monitoring strategy.
UF-6	Scalability considerations	Performance	No discussion of how the system would behave with many users or large data volumes.
UF-7	Backup and recovery	Reliability	No mention of backup, recovery, or disaster recovery strategies.
UF-8	Configuration management	Maintainability	No description of how DB credentials or environment-specific settings are managed.
UF-9	Continuous integration / delivery	Project Management	No mention of CI/CD pipelines or automated build processes.
UF-10	Coding standards and style guidelines	Maintainability	No explicit coding standards or style guidelines are documented.
UF-11	Formal non-functional requirements	Requirements	Non-functional requirements (performance, security, usability) are not formally specified with measurable metrics.

4.4 Checklist Compliance Example: Clear Problem Definition

Feature: Clear problem definition

Description: The problem is clearly defined with appropriate context and scope.

Table 18: Checklist Compliance for “Clear problem definition”

ID	Checklist Item	Status	Explanation
1.1	Problem statement is clear and specific	True	Section 1.1 clearly states that the goal is to develop an application for managing a videogame sales system with catalog, cart, library, statistics, achievements, and subscriptions.
1.2	Context and background are provided	Partial	The statement describes what the system will do but does not explicitly discuss current limitations of existing solutions or stakeholders'needs. Adding a short motivation and context paragraph would improve this.
1.3	Problem scope is well-defined	Partial	Functional scope is described through features, but boundaries (e.g., no real game execution, only simulation; single-user vs multi-user) are not explicitly stated. A “Scope and Limitations” subsection would clarify this.
1.4	Target users and stakeholders are identified	Partial	Users and admins are implicit in use cases, but stakeholders (e.g., platform operators) are not explicitly listed. A short stakeholder list would complete this.

5 Detailed Analysis

5.1 Requirements Quality Assessment

5.1.1 Issues and Recommendations

Requirements needing more detail:

- **REQ-2:** Search criteria are mentioned but not fully specified (e.g., case sensitivity, partial matches). **Recommendation:** Specify supported search operators and expected behavior.
- **REQ-5:** Statistics are recorded but retention period and precision are not defined. **Recommendation:** Define how long statistics are stored and at what granularity.
- **REQ-6:** Achievements are unlocked based on time, but other possible conditions are not discussed. **Recommendation:** Clarify whether only time-based achievements are supported.

- **REQ-7:** Subscription benefits are described qualitatively. **Recommendation:** Explicitly state discount percentages and rules for free games (e.g., number of free games per month).
- **REQ-12:** Maintainability and scalability are goals but not quantified. **Recommendation:** Add maintainability goals (e.g., maximum coupling, test coverage) or scalability assumptions (e.g., expected number of users).

Vague or missing non-functional requirements:

- No explicit performance targets (e.g., maximum response time for catalog search).
- No explicit security requirements (e.g., password hashing, access control policies).
- No usability or accessibility requirements for the CLI.

5.2 Use Case Completeness Analysis

5.2.1 Explicit vs. Implicit Use Cases

All four main use cases are explicitly documented with templates. No additional implicit use cases are strictly required, although one could formalize "User registration" and "User authentication" as separate use cases, which are currently only described in tests and controller descriptions.

5.2.2 Alternative Flow Coverage

- **Strong:** UC-001 and UC-002 have rich alternative flows covering empty cart, insufficient funds, already purchased games, empty library, already installed games, and missing achievements.
- **Moderate:** UC-003 and UC-004 include some alternative flows (invalid credentials, invalid data, insufficient funds, already active subscription) but not all possible error conditions are enumerated.

Recommendations:

- Add explicit use cases or at least templates for user registration and login, since they are central to the platform and heavily tested.
- For UC-003 and UC-004, expand alternative flows to cover all validation errors that are implemented in code (e.g., invalid videogame data, invalid achievement parameters).

5.3 Architecture Quality Evaluation

5.3.1 Design Principles Assessment

Table 19: Architecture Quality Metrics (Qualitative)

Principle	Status	Notes
Separation of Concerns	Good	CLI, controllers, DAOs, and entities are clearly se
Loose Coupling	Good	Controllers depend on DAOs via interfaces/constructors; DB de
High Cohesion	Good	Each controller focuses on a specific area (users, cart, libra
Single Responsibility	Moderate	Some controllers (e.g., CarrelloController) handle several responsibilitie
Error Handling	Moderate	Exceptions are thrown for invalid states; no global error-handling
Security	Weak	Only basic credential checks; no mention of password hashing or auth

Recommendations:

- Document transaction boundaries for operations like purchase and subscription activation to ensure atomicity.
- Introduce a simple error-handling strategy (e.g., mapping exceptions to user-friendly CLI messages) and document it.
- Add a short section on security practices (e.g., password hashing, use of prepared statements, minimal privileges for DB user).

5.4 Test Coverage Analysis

5.4.1 Coverage by Area

- **Business Logic:** Well covered by controller tests, including many negative scenarios.
- **Domain Model:** Carrello is thoroughly tested, especially pricing strategy; other entities are simple and not individually tested.
- **Persistence:** VideogiocoDAO, CarrelloDAO, and LibreriaDAO are tested; other DAOs (UtenteDAO, AbbonamentoDAO, StatisticheVideogiocoDAO, AchievementDAO, AdminDAO) are used indirectly in controller tests but not all have dedicated DAO tests.

Recommendations:

- Add explicit tests for remaining DAOs to ensure full CRUD coverage.
- Add integration tests for some missing alternative flows (e.g., empty cart, already active subscription).
- Consider adding performance-oriented tests (even simple ones) to measure basic operations like catalog search under load.

6 Recommendations

6.1 Priority 1: Critical Items

1. **Formalize Non-Functional Requirements**
   - Add a short section specifying performance, security, and usability requirements (even at a basic level).
   - **Action:** Define simple metrics (e.g., response time for catalog search, password hashing requirement, basic usability goals for CLI).
2. **Complete Test Coverage for Key Alternative Flows**
   - Missing or weakly tested flows: empty cart (UC-001), empty library (UC-002), already active subscription (UC-004), some admin validation errors (UC-003).
   - **Action:** Add JUnit tests that explicitly exercise these scenarios and assert correct error messages or behavior.
3. **Document Security and Transaction Handling**
   - Security and transaction management are implemented implicitly but not documented.
   - **Action:** Add a short subsection describing how credentials are stored, how SQL injection is prevented (prepared statements), and how multi-step operations (e.g., purchase) are kept consistent.

6.2 Priority 2: Important Improvements

1. **Add Explicit Use Cases for Registration and Authentication**
   - These flows are central and already tested but not documented as use cases.
   - **Action:** Create UC templates for “Register user” and “Authenticate user” with preconditions, steps, and alternative flows (e.g., invalid credentials).
2. **Extend DAO Test Coverage**
   - Some DAOs are only indirectly tested.
   - **Action:** Add tests for UtenteDAO, AbbonamentoDAO, StatisticheVideogiocoDAO, AchievementDAO, and AdminDAO similar to VideogiocoDAOTest.
3. **Clarify Scope and Limitations**
   - The system simulates gameplay and uses a CLI; these constraints should be explicit.
   - **Action:** Add a “Scope and Limitations” subsection in the introduction describing what is intentionally out of scope (e.g., real game execution, graphical UI).

6.3 Priority 3: Nice-to-Have Enhancements

1. **Add Deployment and Scalability Notes**
   - Even a simple description (single machine, local PostgreSQL) would improve completeness.
2. **Introduce Basic Logging Strategy**
   - Replace System.out.println with a simple logging framework and document log levels.
3. **Summarize Coding Standards**
   - A short note on naming conventions and code style would strengthen maintainability documentation.

6.4 Summary Checklist

Table 20: Action Item Summary

Priority	**Items**	**Est. Effort**
Critical (P1)	3	1–2 days
Important (P2)	3	2–3 days
Nice-to-Have (P3)	3	2–4 days
Total	**9**	**5–9 days**