Software Engineering Audit Report

[swe]Relazione Pinzani-Davio.pdf

CAPRA

February 25, 2026

Contents

1 Document Context

Project Objective

The application facilitates and optimizes the organization of trains on railway lines, manages personnel, and accounts for cleaning and maintenance of train cars performed at depots or authorized stations. The system generates a model used by staff containing essential information for correct service execution and interfaces with a database through the DAO pattern to ensure fast and secure data management.

Main Use Cases

- UC-1 – Segnala atto vandalico: Train driver reports vandalism acts on the train.
- UC-2 – Richiede pulizia lieve: Train driver requests light cleaning (1 hour duration).
- UC-3 – Richiede pulizia intensiva: Train driver requests intensive cleaning (3 hours duration) following vandalism detection.
- UC-4 – Richiede manutenzione: Train driver requests train maintenance when necessary.
- UC-5 – Conferma pulizia: Trainer confirms or denies cleaning request based on train availability.
- UC-6 – Conferma manutenzione: Trainer confirms or denies maintenance request based on train availability.
- UC-7 – Richiesta modello treno: Trainer requests the system to generate a train model with departure time, line identifier, and direction.
- UC-8 – Errore modello treno: System generates error notification when essential elements are missing (no train driver available or insufficient working hours).

Functional Requirements

- The system shall automatically assign train drivers and trains to lines based on availability and working hour constraints.
- The system shall calculate arrival times at each station based on departure time and travel duration between stations.
- The system shall generate train models containing unique train code, train driver data, station list with calculated times, and train identifier.
- The system shall manage maintenance requests: train drivers request maintenance, trainers approve or deny based on substitute train availability.
- The system shall manage cleaning requests: light cleaning (1 hour) or intensive cleaning (3 hours following vandalism reports).
- The system shall send notifications to trainers when train drivers request maintenance or cleaning.
- The system shall allow trainers to modify train models with higher database access level than train drivers.
- The system shall allow only train drivers to report vandalism acts on trains.
- The system shall generate warning messages when no available train driver exists at departure station or when working hours would be exceeded.
- The system shall persist and manage data through DAO pattern: personnel information, daily train models, train identifiers and characteristics, and station data.

Non-Functional Requirements

- The system shall provide fast and secure database access through DAO pattern implementation.
- The system shall handle multiple railway lines efficiently.
- The system shall support scalability for future expansion with multiple trainers collaborating on train model management.
- Train models shall be created daily and deleted at end of day.
- Maintenance operations shall require one day of train unavailability.

Architecture

The system follows a layered architecture with DAO (Data Access Object) pattern for database integration. The application separates concerns between user interface interactions (train drivers and trainers), business logic for train model generation and resource allocation, and data persistence layer. The system manages three main entities: personnel (train drivers and trainers), trains (convoys with maintenance and cleaning status), and railway lines (with stations and schedules). The trainer role has elevated database access permissions for model modifications, while train drivers have read access and can report operational issues. The architecture supports asynchronous notifications between train drivers and trainers for maintenance and cleaning requests.

Testing Strategy

The document does not provide explicit testing strategy details. However, based on the use case templates and requirements, testing would involve: verification of train model generation with correct arrival time calculations, validation of train driver and train assignment respecting working hour constraints, testing notification delivery for maintenance and cleaning requests, verification of trainer approval/denial logic based on substitute train availability, and error handling validation when required personnel or trains are unavailable. Integration testing would verify DAO pattern implementation for secure and reliable database operations.

2 Executive Summary

Quick Overview

Total issues: **14** — HIGH: **5** MEDIUM: **7** LOW: **2**

Average confidence: **99%**

Executive Summary: Audit of Software Engineering Document

This document is a requirements specification for a train scheduling and maintenance management system, written as a university Software Engineering project. It presents a set of use cases, actors, and business rules intended to guide the design and implementation of the system. The primary purpose is to formally capture functional and non-functional requirements through use case diagrams, textual descriptions, and detailed UC templates.

The audit identified **14 issues across four categories**, with 5 classified as HIGH severity. The dominant patterns are: (1) **incomplete and inconsistent use case specifications**, particularly for maintenance and cleaning requests, where templates are omitted or marked as "similar" without full documentation; (2) **underspecified error handling and system-initiated interactions**, such as notification flows and error model generation conditions, which lack explicit UC templates or clear alternative flows; (3) **weak post-conditions and state management**, where important system state changes implied by normal flows are not captured in post-conditions, limiting the ability to reason about correctness; and (4) **complete absence of a testing strategy**, with no test cases, test plans, or traceability between requirements and verification activities.

The **critical areas (HIGH severity)** are: (1) missing full UC templates for "richiesta manutenzione" and "pulizia lieve" (ISS-002), which creates ambiguity on preconditions, postconditions, and flows; (2) unclear error handling for model generation, with informal issue lists not integrated into alternative flows

or linked to the error UC (ISS-003); (3) no testing strategy or test cases whatsoever (TST-001), leaving all requirements unverified; (4) no tests for normal flows of key use cases such as notification generation and resource assignment (TST-002); and (5) no tests for critical error conditions related to resource availability and working hour constraints (TST-003).

Overall Quality Assessment: The document demonstrates a reasonable understanding of use case modeling and captures the main actors and workflows. However, it suffers from significant gaps in completeness, consistency, and rigor. The requirements are partially specified, with several use cases either omitted or underspecified. The absence of any testing strategy is a critical deficiency that prevents validation of the requirements and leaves the document unsuitable for development or acceptance testing. The document requires substantial revision before it can serve as a reliable basis for implementation.

Priority Actions:

1. **Complete all use case templates** with full preconditions, postconditions, normal flows, and alternative flows for "richiesta manutenzione", "pulizia lieve", and "Errore modello treno". Ensure that error conditions are explicitly modeled as alternative flows or separate UCs, and that all system-initiated interactions (notifications, status updates) are documented with clear triggers and outcomes.
2. **Develop a comprehensive testing strategy** that includes: (a) a test plan linking each UC to one or more test cases; (b) test cases for all normal flows, covering notification generation, resource assignment, and model generation; (c) test cases for all critical error conditions (capotreno unavailability, working hour violations); and (d) test cases for alternative flows (e.g., refusal of cleaning or maintenance when no substitute train is available).
3. **Reconcile inconsistencies** between textual descriptions and diagrams, particularly regarding actor roles in maintenance requests (capotreno vs. formatore), the timing and conditions for model generation (beginning of day only vs. multiple times per day), and the scope of essential elements that can trigger error conditions. Update diagrams and templates to reflect a single, consistent interpretation of each business rule.

3 Strengths

- **Comprehensive Use Case Coverage:** The document provides well-structured use case diagrams for both primary actors (Capotreno and Formatore), clearly depicting system interactions and use case relationships including include and extend stereotypes. This demonstrates thorough requirements elicitation.
- **Detailed Use Case Templates:** The project includes formal use case templates with preconditions, postconditions, normal flows, and alternative flows (e.g., Tabella 2 showing alternative scenarios for Conferma pulizia). This level of specification supports clear requirements documentation.
- **Well-Defined System Responsibilities:** The Statement section (1.2) explicitly outlines automated operations and system functions, clearly delineating responsibilities between actors (Capotreno, Formatore) and the system, including database integration via the DAO pattern.
- **Appropriate Design Pattern Selection:** The document demonstrates awareness of architectural best practices by specifying the DAO (Data Access Object) pattern for database integration, ensuring data security and accessibility while maintaining separation of concerns.
- **Error Handling Specification:** The project includes explicit error scenarios (Tabella 4: Errore modello treno) documenting system behavior when essential elements are missing, such as unavailable personnel or insufficient working hours.
- **Future Enhancement Planning:** Section 1.3 identifies potential improvements including scalability considerations, personnel scheduling refinements, and multi-formatore collaboration, demonstrating forward-thinking software engineering practices.

4 Expected Feature Coverage

Of 7 expected features: 2 present, 3 partial, 2 absent. Average coverage: **46%**.

Feature	Status	Coverage	Evidence
Unit testing framework implementation	Absent	0% (0/5)	The document does not mention any tests, unit testing framework, assertions, test classes, mocks, or coverage of unit/integration testing.
Use of UML Diagrams for system modeling	Partial	50% (4/8)	Two use case diagrams are explicitly described (for Capotreno and Formatore), using standard UML notation such as actors, use cases, and «include»/«extend» relationships, and they clarify functional requirements by showing interactions. There is no mention of class diagrams, other structure diagrams, UI mockups, or explicit visualization of navigation flows.
Identification and definition of system actors	Present	100% (8/8)	The document clearly identifies actors such as Capotreno, Formatore, and Sistema, and describes their responsibilities (e.g., Capotreno requests maintenance and cleaning, signals vandalism; Formatore generates modello treno, confirms or denies maintenance/pulizia). Use case templates and statements define interactions, specific user actions, and functional requirements per role, showing a structured approach to user requirements and delineation of system functionalities.
Definition and Documentation of Use Cases	Present	100% (5/5)	Multiple use cases are documented with templates such as 'Richiesta pulizia intensiva', 'Conferma pulizia', 'Richiesta modello treno', and 'Errore modello treno'. Each includes user interactions and goals, detailed normal and alternative flows, and explicit pre-conditions and post-conditions. Relationships between use cases are shown in the diagrams via «include» and «extend» (e.g., 'Segnala atto vandalico' includes 'Richiede pulizia intensiva', 'Conferma manutenzione' extends 'Richiede manutenzione').

Feature	Status	Coverage	Evidence
User interface and interaction design principles	Partial	14% (1/7)	The document defines distinct user roles and their functionalities (Capotreno and Formatore) and describes some interaction steps (e.g., Capotreno uses a button to request pulizia intensiva), but there are no UI mockups, no explicit navigation structures, no reservation process, no detailed input validation mechanisms, and no description of dynamic UI updates.
Separation of concerns in software architecture	Absent	20% (1/5)	The document mentions that the application interfaces with a database through the DAO pattern, hinting at some separation between data access and other logic, but it does not define packages for Model/View/Controller/Service, does not describe controller–model or controller–service–DAO relationships, and does not clearly describe architectural roles or modular design principles.
Data Access Object (DAO) pattern	Partial	43% (3/7)	The statement section explicitly says the application interfaces with a database through the DAO pattern to save and manage data, and later notes that integration with the database will be done via DAO to store personnel, modelli treno, convogli, and stazioni data, documenting the pattern for managing data access and its abstraction role. However, there is no detail on CRUD operations, specific DAO interfaces or classes, separate DAOs per entity, or testing strategies for data access.

5 Summary Table

Category	HIGH	MEDIUM	LOW	Total
Architecture	0	0	1	1
Requirements	2	5	1	8
Testing	3	2	0	5
Total	**5**	**7**	**2**	**14**

6 Issue Details

6.1 Architecture (1 issues)

ISS-001 — LOW [100%] — Page 2 The document declares the use of the DAO pattern for database interaction, but there is no later description of concrete classes, interfaces, or how tests will isolate DAO from business logic. This makes it hard to verify architectural consistency and to design tests that respect the intended separation of concerns.

> *Inoltre l'applicazione si interfaccia con un database attraverso il pattern DAO per poter salvare e gestire i dati garantendo un servizio rapido e sicuro.*

Recommendation: Add a brief architectural section that lists the main DAO interfaces/classes (e.g., ConvoglioDAO, CapotrenoDAO, ModelloTrenoDAO) and the service or controller classes that use them. Then, in the testing section, explain how you will test business logic independently of the DAO (e.g., by mocking DAO interfaces in unit tests) and, if applicable, how you will test DAO implementations separately (e.g., simple integration tests against a test database). Even a simple class diagram or textual list will improve consistency between the declared DAO pattern and the actual implementation and tests.

6.2 Requirements (8 issues)

ISS-002 — HIGH [100%] — Page 5 Systematic omission of full templates for key use cases ("richiesta manutenzione" and "pulizia lieve"). They are only stated as "similar" to another UC, so their specific pre-conditions, post-conditions, normal and alternative flows, and any constraints are undocumented. This creates ambiguity on when they can be invoked, what data is required, and how they differ from the intensive cleaning request, which directly impacts correctness and testability.

> *Gli use case templates dei casi richiesta manutenzione e pulizia lieve non vengono mostrati in quanto il loro funzioamento 'e molto simile al caso appena descritto e si concludono tutti con l'invio di una notifica al formatore.*

Recommendation: Create complete use case templates for both "Richiesta manutenzione" and "Richiesta pulizia lieve" instead of referring to similarity. For each of these UCs, explicitly specify: (1) Description and actors; (2) Pre-conditions (e.g., treno ha finito il servizio? presenza in una certa stazione? segnalazioni particolari?); (3) Post-conditions (e.g., stato del treno, stato della richiesta, notifica generata); (4) Normal flow steps, including how the capotreno interagisce con il sistema; (5) Alternative/error flows (es. richiesta non valida, treno già in manutenzione/pulizia, richiesta duplicata). Make clear in what they differ from "Richiesta pulizia intensiva" (durata, condizioni di attivazione, vincoli di stazione, ecc.) so that tests can be derived unambiguously.

ISS-003 — HIGH [100%] — Page 7 The error conditions for model generation are only informally listed as "Issues" in UC "Richiesta modello treno" and are not integrated as alternative flows or linked explicitly to UC "Errore modello treno". This makes the business rule about crew availability and working hours unclear: it is not specified whether the system should always trigger UC "Errore modello treno" in these cases, whether there are fallback strategies, or how the user is guided to resolve the issue.

> *Issues Possibile assenza del capotreno nella stazione di par- tenza o con disponibilit'a di ore in modo da non sfo- rare l'orario lavorativo*

Recommendation: Refactor the handling of missing/overworked capotreno into explicit alternative flows in UC "Richiesta modello treno" and clearly connect them to UC "Errore modello treno". For example: add an "Alternative flow" section to "Richiesta modello treno" with steps like: (2a) Il sistema non trova nessun capotreno in stazione di partenza → include UC "Errore modello treno"; (2b) Il sistema trova capotreno ma nessuno con ore disponibili entro il limite → include UC "Errore modello treno". Move the current "Issues" text into these structured flows and specify what the formatore sees (messaggio, contenuto, possibili azioni successive). This will make the business rule testable and consistent with the error UC.

ISS-004 — MEDIUM [100%] — Page 3 The business rule about who can request and who can confirm maintenance is described textually but is not consistently reflected in the use case diagrams and templates. In the diagrams, "Richiede manutenzione" appears associated with the formatore (via extend from "Conferma manutenzione"), while the text states that the capotreno decides and notifies the formatore. There is no explicit UC template for the capotreno's maintenance request, and the actor roles in the formatore's UCs do not clearly show this interaction.

> *Il capotreno notifica il formatore nel momento in cui decide che il convo- glio deve andare in manutenzione, tuttavia il formatore pu'o decidere se il convoglio ha l'abilitazione o meno di andare in officina.*

Recommendation: Align the maintenance workflow across text, diagrams, and templates. Introduce a dedicated UC template for "Richiesta manutenzione" by the capotreno, with Capotreno as primary actor and Formatore as secondary (receiver of the notification). In the formatore diagram, ensure that "Richiede manutenzione" is either removed (if it is not an action of the formatore) or renamed to something like "Gestisce richiesta manutenzione". In the "Conferma manutenzione" template, explicitly reference the prior capotreno request in the pre-conditions and describe how the notification is received. This will remove ambiguity about responsibilities and make the flow testable end-to-end.

ISS-005 — MEDIUM [100%] — Page 4 The use case diagrams and templates do not consistently model system-initiated interactions and notifications that are described in the text. For example, the statement mentions that the capotreno notifies the formatore and that the system sends notifications for cleaning and maintenance, but there is no explicit UC or flow describing how the formatore "riceve" these notifications (beyond a single step in "Conferma pulizia"). Similarly, "Riceve errore modello treno" appears in the diagram but has no corresponding template, leaving the behavior of this system-initiated interaction underspecified.

> *Notiamo come attraverso questi diagrammi vengono messe in risalto le operazioni dirette tra utente e sistema ma anche eventuali interazioni che partono dal sistema e che l'utente potrebbe aspettare di ricevere.*

Recommendation: For all system-initiated interactions mentioned in the text (notifications to formatore, error messages, etc.), add or complete corresponding UCs or flows. Concretely: (1) Add a UC template for "Riceve errore modello treno" or merge it into "Errore modello treno" with explicit steps describing how the formatore sees and acknowledges the error; (2) In the cleaning and maintenance workflows, add explicit steps or separate UCs for "Ricezione notifica" by the formatore, including pre-conditions (notifica esistente), post-conditions (stato della richiesta aggiornato), and possible alternative flows (notifica non leggibile, richiesta già gestita). Ensure the diagrams show these UCs and that the actor associations match the templates.

ISS-006 — MEDIUM [100%] — Page 6 Post-conditions across several UCs are too generic and do not capture important state changes implied by the normal flow. For "Richiesta modello

treno", the flow states that the system assigns capotreno and convoglio and makes them unavailable, but the post-condition only mentions returning the model to the formatore. Similar simplifications appear in other UCs (e.g., "Conferma pulizia" only states that the train is sent to cleaning or put back in service, without specifying status changes or constraints on future requests). This weakens the ability to reason about system state and derive precise tests.

> *Post-conditions Il sistema ritorna al formatore il modello treno.*

Recommendation: Strengthen post-conditions for all main UCs to reflect key state changes. For "Richiesta modello treno", extend the post-condition to include: (a) esiste un nuovo modello treno persistito nel sistema; (b) il capotreno e il convoglio selezionati sono marcati come non disponibili per il tempo assoluto di percorrenza; (c) la linea ha un servizio assegnato per quella fascia oraria. For "Conferma pulizia" and the analogous maintenance UC, specify the resulting status of the train (e.g., in_pulizia, in_manutenzione, in_servizio) and any constraints on further requests. Review each UC template and ensure that the post-condition summarizes the final observable state implied by the normal/alternative flows, not just the last UI action.

ISS-007 — MEDIUM [90%] — Page 6 The pre-condition for "Richiesta modello treno" states that it can only be executed at the beginning of the working day with no trains assigned to lines. This conflicts with the more general description in the Statement, where the formatore can generate models to guarantee efficient service, and with the idea of multiple models per day. It is unclear whether the system supports generating additional models later in the day (e.g., for extra services or substitutions) or if this is intentionally forbidden.

> *Pre-conditions Inizio della giornata lavorativa, quindi assenza di tre- ni assegnati alle linee*

Recommendation: Decide and document clearly the intended lifecycle of "modello treno" generation. If models can only be generated once at the start of the day, explicitly state this constraint also in the Statement and explain how extraordinary services are handled. If models can be generated multiple times during the day, relax or refine the pre-condition to something like "non esiste già un modello per quella linea e fascia oraria" and describe how the system behaves when some trains are already assigned. Update both the UC template and the introductory requirements so they are consistent, and add alternative flows for attempts to generate a model when the pre-condition is not met.

ISS-008 — MEDIUM [100%] — Page 8 UC "Errore modello treno" is underspecified and partially inconsistent with the rest of the document. The description generically mentions "element essenziali" but the normal and alternative flows only cover the absence or unavailability of capotreno. Other essential elements mentioned in the statement (convoglio, linea, orari, stazioni) are not covered, so it is unclear whether they can also trigger this UC and what the system should do in those cases.

> *UC Errore modello treno Level System goal Description Il sistema non trova element essenziali e genera un errore*

Recommendation: Clarify and align UC "Errore modello treno" with the business rules in the Statement. Either: (a) restrict the UC explicitly to crew-related errors (rename description to something like "assenza di capotreno o ore disponibili" and adjust text accordingly), or (b) extend the normal/alternative flows to cover all essential elements: missing convoglio, linea non valida, stazioni mancanti, dati orari inconsistenti, etc. For each error type, specify: the condition, the generated message content (at least at a high level), and the post-condition (e.g., nessun modello creato, nessuna assegnazione effettuata). This will make the UC coherent and allow you to derive comprehensive tests.

ISS-009 — LOW [95%] — Page 2 Non-functional requirements such as "servizio rapido e sicuro" are stated qualitatively but are not translated into measurable or verifiable criteria in the requirements or UCs. There are no performance constraints (e.g., maximum time to generate a modello treno) or security requirements (e.g., access control rules beyond a brief mention of different access levels) that could be tested or validated.

> *Inoltre l'applicazione si interfaccia con un database attraverso il pattern DAO per poter salvare e gestire i dati garantendo un servizio rapido e sicuro.*

Recommendation: Refine the non-functional requirements into concrete, testable statements and ensure they are consistent with the functional UCs. For example: (1) Define a performance requirement for "Richiesta modello treno" (e.g., il modello deve essere generato entro X secondi per una linea tipica); (2) Specify security/access rules, such as "solo il formatore può modificare un modello treno" and "il capotreno può solo creare richieste di pulizia/manutenzione" and reference these as pre-conditions or constraints in the relevant UCs; (3) If data safety is important, add requirements about persistence (e.g., modelli treno non devono essere persi in caso di crash durante la giornata). This will make the "rapido e sicuro" claim meaningful and gradable.

6.3 Testing (5 issues)

TST-001 — HIGH [100%] — Page 4 The document defines several use cases and detailed templates, but there is no section describing any testing strategy, test cases, or how these use cases will be verified. This creates a complete lack of traceability between requirements/use cases and concrete tests.

> *Per capire al meglio il funzionamento del software andiamo ad evidenziare le varie interazioni tra sistema e utenti. Per poter fare ci'o utilizziamo gli use case diagrams.*

Recommendation: Add a dedicated testing section that, for each main UC (e.g., "Richiesta pulizia intensiva", "Conferma pulizia", "Richiesta modello treno", "Errore modello treno"), defines at least one test case. For each test, explicitly reference the UC ID/name, describe input data, pre-conditions, expected system behavior, and post-conditions. A simple table per UC (UC name, test case ID, description, steps, expected result) is sufficient and will create clear traceability between requirements and tests.

TST-002 — HIGH [100%] — Page 5 Normal flows of key use cases are specified, but there are no corresponding tests described to verify that the system correctly executes these normal flows (e.g., notification generation and sending, assignment of convoglio and capotreno, generation of modello treno). This is a systemic gap across all main UCs.

> *Normal flow 1. Il treno conclude il servizio su una linea 2. Il capotreno rileva e segnala un atto vandalico 3. Il capotreno richiede la pulizia intensiva del con- voglio attraverso un pulsante 4. Il sistema genera la notifica di pulizia intensiva 5. Il sistema invia la notifica al formatore*

Recommendation: For each UC normal flow ("Richiesta pulizia intensiva", "Conferma pulizia", "Richiesta modello treno"), define at least one positive test case that follows the numbered steps. For example, for "Richiesta pulizia intensiva" create a test like: given a treno that has concluso il servizio, when the capotreno presses the pulizia intensiva button, then the system must create a notifica with the correct attributes and mark it as inviata al formatore. Implement these as unit tests or integration tests depending on your architecture, but document them explicitly in the report with clear mapping to each step of the normal flow.

TST-003 — HIGH [100%] — Page 7 Critical error conditions related to resource availability (assenza del capotreno, superamento orario lavorativo) are identified in the requirements, but there are no tests described to verify that the system correctly detects these conditions and reacts as specified (e.g., by generating an error message). This affects safety and correctness of scheduling logic.

> *Issues Possibile assenza del capotreno nella stazione di par- tenza o con disponibilit'a di ore in modo da non sfo- rare l'orario lavorativo*

Recommendation: For all use cases where resource availability is an issue ("Richiesta modello treno" and "Errore modello treno"), add explicit negative test cases. Examples: (1) No capotreno in stazione di partenza: verify that the system does not assign a modello treno and instead produces the expected errore. (2) Capotreno presente ma con ore che superano il massimo: verify that the system rejects the assignment and generates the correct errore. Document these tests with clear input data (personale list, orari, stazione), expected error messages, and link them to the "Issues" and "Errore modello treno" UCs.

TST-004 — MEDIUM [100%] — Page 6 Alternative flows (branching behavior) are specified for several use cases, but there is no indication of tests that verify these alternative paths, such as refusal of pulizia or manutenzione when no treno sostitutivo is available. This is a systemic lack of coverage for alternative flows.

> *Alternative flow 2a. Il formatore osserva l'assenza di un treno sosti- tutivo per la linea 3a. Il formatore rifiuta la pulizia 4a. Il sistema rimette il treno in servizio sulla linea*

Recommendation: For each UC that defines an "Alternative flow" (e.g., "Conferma pulizia" and the analogous "Conferma manutenzione"), add at least one dedicated test case that forces the alternative path. For "Conferma pulizia", create a test where the system state has no available treno sostitutivo: simulate the formatore's decision to rifiutare and assert that the treno status is set back to "in servizio" and no pulizia is scheduled. Clearly label these tests (e.g., testConfermaPulizia_SenzaTrenoSostitutivo_Rifiuto) and reference the specific alternative flow steps (2a–4a) in the test description.

TST-005 — MEDIUM [100%] — Page 8 The UC "Errore modello treno" defines a system-level error handling behavior, but there is no description of tests that verify the notification mechanism to the formatore (content, timing, and conditions of the errore). This leaves a critical error-reporting path untested.

> *UC Errore modello treno Level System goal Description Il sistema non trova element essenziali e genera un errore Actors • Sistema (primary) • Formatore (secondary) Pre-conditions Il formatore ha fatto richiesta di generare il modello treno Post-conditions Il sistema notifica il formatore dell'errore*

Recommendation: Design and document tests specifically for "Errore modello treno" that verify: (1) when essential elements (e.g., capotreno, convoglio) are missing, the system creates an errore object/message; (2) the errore is actually delivered to the formatore (e.g., via UI message, log entry, or notification entity); (3) the errore content matches the missing condition. Include at least one test for the main flow (no capotreno in stazione) and one for the alternative flow (capotreno presente ma senza ore disponibili), and explicitly state in the test descriptions that they cover UC "Errore modello treno".

7 Priority Recommendations

The following actions are considered priority:

1. **ISS-002** (p. 5): Create complete use case templates for both "Richiesta manutenzione" and "Richiesta pulizia lieve" instead of referring to similarity.
2. **ISS-003** (p. 7): Refactor the handling of missing/overworked capotreno into explicit alternative flows in UC "Richiesta modello treno" and clearly connect them to UC "...
3. **TST-001** (p. 4): Add a dedicated testing section that, for each main UC (e.g., "Richiesta pulizia intensiva", "Conferma pulizia", "Richiesta modello treno", "Errore mo...
4. **TST-002** (p. 5): For each UC normal flow ("Richiesta pulizia intensiva", "Conferma pulizia", "Richiesta modello treno"), define at least one positive test case that fo...
5. **TST-003** (p. 7): For all use cases where resource availability is an issue ("Richiesta modello treno" and "Errore modello treno"), add explicit negative test cases.

8 Traceability Matrix

Of 12 traced use cases: 0 fully covered, 11 without design, 12 without test.

ID	Use Case	Design	Test	Gap
UC-1	Richiede pulizia intensiva	×	×	No design/architecture elements or test cases are described for this use case.
UC-2	Richiede pulizia lieve	×	×	Use case mentioned as similar to 'Richiesta pulizia intensiva' but no design or tests are provided.
UC-3	Richiede manutenzione	×	×	Use case referenced in text and diagrams but lacks explicit design mapping and test coverage.
UC-4	Segnala atto vandalico	×	×	Use case appears in the use case diagram but no corresponding design or tests are documented.
UC-5	Richiede pulizia	×	×	Generic 'Richiede pulizia' use case for formatore is shown in the diagram but has no detailed design or tests.
UC-6	Conferma pulizia	×	×	Template is provided for this use case but no implementation design or test cases are specified.
UC-7	Richiede manutenzione (formatore)	×	×	Use case appears in the formatore diagram but lacks detailed template, design description, and tests.

ID	Use Case	Design	Test	Gap
UC-8	Conferma manutenzione	×	×	Mentioned as similar to 'Conferma pulizia' but no explicit design artifacts or test cases are documented.
UC-9	Richiesta modello treno	×	×	Detailed template exists but there is no mapping to concrete classes/components or to test cases.
UC-10	Errore modello treno / Riceve errore modello treno	×	×	Error-handling use case is specified in a template and diagram but has no associated design or tests.
UC-11	Integrazione con database tramite pattern DAO	✓	×	High-level design choice (DAO pattern) is described but no specific DAO classes or database integration tests are define...
UC-12	Gestione livelli di accesso al database (Formatore vs Capotreno)	×	×	Requirement on differentiated access levels is stated but no access-control design or tests are provided.

9 Terminological Consistency

Found **10** terminological inconsistencies (2 major, 8 minor).

Group	Variants found	Severity	Suggestion
Section 1.2 naming	"Statement"; "statement"; "enunciato nel paragrafo 1.2"; "elencate negli statement"	MINOR	Use a single language for the section title and its references, preferably Italian "Statement" -> "Requisiti" or English consistently throughout.
Use case diagram naming	"Use Case Diagram"; "use case diagrams"; "diagrammi dei casi d'uso"	MINOR	Use a single language for the term, preferably Italian "diagrammi dei casi d'uso" or English "use case diagrams" consistently.
Use case template naming	"Templates"; "use case templates"; "templates dei casi d'uso"	MINOR	Use a single language for the term, preferably Italian "template dei casi d'uso" or English "use case templates" consistently.
DAO pattern naming	"pattern DAO"; "DAO pattern" (implicit English form in an otherwise Italian sentence)	MINOR	Use a single language for the term, e.g. consistently "pattern DAO" or fully English "DAO pattern".

Group	Variants found	Severity	Suggestion
Figure captions vs headings for diagrams	"Use Case Diagram"; "Use case diagram"; "Use case diagram 1"; "Use case diagram 2"; "diagrammi"	MINOR	Use a single language for the term, e.g. consistently Italian "diagramma dei casi d'uso" or English "Use Case Diagram".
Actor "capotreno" naming	"capotreno"; "capotrteno"	MAJOR	Use a single, consistently spelled term for the same actor, preferably the correct Italian railway role name (likely "capotreno").
Use case naming in text vs UC label	"UC"; "casi d'uso"; "use case"	MINOR	Use a single language for the term, e.g. consistently Italian "caso d'uso" or English "use case".
Undefined technical term DAO	"pattern DAO"	MAJOR	Define the technical term when first introduced, e.g. briefly explain what "pattern DAO" means in the context of the system.
System naming	"sistema"; "System goal"; "Sistema" (as actor in UC tables)	MINOR	Use a single language for the term, e.g. consistently Italian "sistema" or English "system" in headings and UC tables.
UC template field labels language	"Level"; "User goal"; "System goal"; other Italian labels in the same tables	MINOR	Use a single language for the term, e.g. consistently Italian "livello" or English "Level" in UC templates.