Software Engineering Audit Report

relazione_florea_corretta.pdf

CAPRA

February 25, 2026

Contents

1 Document Context

Project Objective

The application is a heterogeneous sensor management system developed in Java that integrates three types of sensors with different interfaces (A, B, C) into a unified system. It enables coherent measurement management and automatic notification of a control unit (centralina) when sensor values change, using design patterns Adapter, Composite, and Observer.

Main Use Cases

- UC-1 – Visualizza misure correnti: User requests to view the latest known state of measurements from all sensors managed by the system.
- UC-2 – Richiedi aggiornamento specifico: User requests a new measurement from a specific sensor and displays the updated state.
- UC-3 – Richiedi aggiornamento globale: User requests new measurements from all managed sensors and displays the complete updated state.
- UC-4 – Presenta elenco misure: System presents to the user the list of current measurements (included in UC-1, UC-2, UC-3).

Functional Requirements

- RF1 (Acquisizione dati): System must obtain measured values from each sensor type (A, B, C) independently of its specific interface.
- RF2 (Adattamento obbligatorio): Original sensor classes (A, B, C) are non-modifiable external components; system must adapt them to a common internal interface without altering original classes.
- RF3 (Aggregazione sensori): System must allow creation of sensor groups and retrieval of all measurements from a group via a single operation on the root element.
- RF4 (Notifica centralina): System must include a control unit automatically notified when sensor updates are forced, containing sensor identification and measured value.
- RF5 (Collaudo unitario): JUnit unit tests must be implemented to verify correct functionality.

Non-Functional Requirements

- The system must maintain separation of concerns across four packages: `domain`, `data`, `observer`, and `sensors`.
- Original sensor classes must remain unmodified and isolated from the rest of the system.
- The system must support hierarchical composition of sensors with uniform treatment of individual sensors and groups.
- Data transfer between components must use standardized format (Data Transfer Object pattern).

Architecture

The system is structured in four packages: (1) `domain` implements Composite pattern (Component interface, Composite class) and Adapter pattern (AdapterA/B/C classes extending Observable); (2) `data` contains the immutable `Misurazione` data transfer object for standardized data representation; (3) `observer` contains the `Centralina` class implementing the Observer interface to receive notifications; (4) `sensors` contains original sensor classes A, B, C with different measurement methods (getMeasure(), misura(), measure()). The architecture combines three design patterns: Adapter (to unify heterogeneous sensor interfaces), Composite (to organize sensors hierarchically), and Observer (to notify the control unit of measurement changes). A command-line interface enables user interaction with the system.

Testing Strategy

Unit tests using JUnit 5 verify: (1) Adapter tests (AdapterATest, AdapterBTest, AdapterCTest) check interface implementation, measurement retrieval delegation, and observer notification using mock sensors and test observers; (2) Composite tests verify child management (add/remove), measurement aggregation from all children, and notification propagation; (3) Centralina tests verify correct notification reception, measurement storage and updates, and multi-sensor handling; (4) Sensor tests (SensoreATest, SensoreBTest, SensoreCTest) verify initialization and measurement value logic; (5) Misurazione tests verify correct data storage and output formatting. All tests executed successfully as shown in the test execution report.

2 Executive Summary

Quick Overview

Total issues: **14** — HIGH: **1** MEDIUM: **10** LOW: **3**

Average confidence: **97%**

Executive Summary: Audit Report on Software Engineering Document

This document is a Software Engineering report submitted by a university student describing the design and implementation of a measurement acquisition system with a composite sensor architecture and observer-based notification pattern. The report includes functional requirements, use cases, architectural design, sequence diagrams, and unit test specifications. Its purpose is to demonstrate compliance with design patterns (Composite, Adapter, Observer) and to validate functional requirements through structured testing.

The audit identified **14 issues across four categories**. The predominant pattern is **incomplete specification and traceability gaps**: functional requirements lack observable success criteria, preconditions, and postconditions; use cases are described with minimal alternative flows and no error handling details; and critical design decisions (such as the distinction between pull-based and push-based measurement retrieval) are left ambiguous. Additionally, the mapping between requirements, use cases, architectural components, and test scenarios is either missing or implicit, making it difficult to verify that the system actually satisfies its stated goals.

The **critical area of concern (HIGH severity)** is the absence of end-to-end traceability from functional requirements to test coverage. Specifically, **TST-001** reveals that unit tests are organized by class but do not demonstrate that the complete functional flows described in RF1 and RF3 (measurement acquisition and aggregation through the Composite and Adapter layers) are validated as integrated scenarios. This gap undermines confidence that the core business logic is actually verified. Additionally, several medium-severity issues in the Requirements category (ISS-002, ISS-003, ISS-004, ISS-005, ISS-006, ISS-007) indicate that use cases lack sufficient detail on error conditions, boundary cases, and state guarantees, which compounds the testing problem.

Overall quality assessment: The document demonstrates a reasonable understanding of design patterns and provides a structured approach to the problem. However, it suffers from incomplete specification, weak traceability, and insufficient test coverage of functional requirements. The work is **below the expected standard** for a professional Software Engineering deliverable and requires substantial revision before it can serve as a reliable basis for implementation or verification.

Three priority actions:

1. **Establish end-to-end traceability**: Create an explicit mapping table linking each functional requirement (RF1–RF5) and use case to specific test scenarios (unit, integration, or system level). Add at least one integration test per use case that exercises the complete flow from user input through the Composite/Adapter/Observer chain to output, covering both happy paths and the documented alternative flows.

2. **Enhance use case specifications**: Revise all use cases to include explicit preconditions (e.g., "at least one sensor is registered"), postconditions (e.g., "Centralina's internal measurement list is updated"), and at least two alternative flows per use case covering error conditions (invalid input, missing data, sensor failure). Clarify whether measurement retrieval is pull-based (via Composite.ottieniMisura()) or push-based (via Observer notifications), and document this design decision in the architectural section.
3. **Expand test coverage to include negative scenarios and the CLI layer**: Add unit or integration tests that explicitly verify alternative flows (no measurements available, invalid sensor identifier, invalid menu input) and the behavior of the command-line interface layer shown in the activity diagram. Ensure that tests validate not only class-level behavior but also the interaction between the CLI controller, domain objects, and error handling as described in the use cases.

3 Strengths

- **Well-organized multi-package architecture with clear separation of concerns.** The system is divided into four distinct packages (domain, data, observer, sensors) with clearly defined responsibilities. Each package maintains independence and low coupling, facilitating maintainability and future extensibility.
- **Correct application of three design patterns.** The document demonstrates proper implementation of Adapter (AdapterA/B/C to handle heterogeneous sensor interfaces), Composite (hierarchical sensor grouping with uniform treatment), and Observer (automatic notification mechanism) patterns, with clear motivation and implementation details provided for each.
- **Comprehensive requirements and use case documentation.** Five functional requirements (RF1-RF5) are explicitly defined and traced through detailed use case descriptions with base and alternative flows, providing clear traceability between requirements and system behavior.
- **Complete UML documentation with multiple diagram types.** The document includes use case diagrams, class diagrams with package details, sequence diagrams illustrating data flow, and activity diagrams showing interactive workflows, providing thorough visual specification of the system design.
- **Systematic unit testing coverage.** The implementation includes JUnit tests for all major components (adapters, Composite, Observer, sensors, and data transfer objects), with evidence of successful test execution shown in the document.

4 Expected Feature Coverage

Of 7 expected features: 2 present, 3 partial, 2 absent. Average coverage: **53%**.

Feature	Status	Coverage	Evidence
Unit testing framework implementation	Present	80% (4/5)	The document describes JUnit 5 unit tests with assertions (e.g., assertEquals, assertNotNull, assertInstanceOf in AdapterATest), explains a systematic approach to testing each component (sections 4.1–4.5), defines dedicated test classes such as AdapterATest, CompositeTest, CentralinaTest, SensoreATest, MisurazioneTest, and shows isolated testing using mocks (MockSensoreA, TestObserver, TestComponent, TestObservable). It does not mention any explicit integration testing beyond unit scope.
Use of UML Diagrams for system modeling	Present	88% (7/8)	The report includes a use case diagram with actor Utente and use cases (Visualizza misure correnti, Richiedi aggiornamento specifico, Richiedi aggiornamento globale, Presenta elenco misure) and their relationships, multiple class diagrams for packages (sensors, domain, observer, data, java.util), and an overall class diagram showing component relationships. It uses standardized UML notation and clarifies functional requirements via diagrams. It also provides an activity diagram that visualizes navigation/interaction flow. However, there are no UI mockups; the only UI-related artifact is the activity diagram for the CLI.
Identification and definition of system actors	Partial	75% (6/8)	The document clearly identifies the user role Utente and describes its responsibilities and actions (visualizzare le misure, richiedere aggiornamento specifico, richiedere aggiornamento globale) in the use case section. It defines interactions between Utente and the system components (centralina, adapters/composite) via the use case diagram, sequence diagram, and activity diagram, and presents functional requirements RF1–RF5 that are driven by user needs. However, there is only one user role, so there are no differentiated functional requirements per multiple roles.

Feature	Status	Coverage	Evidence
Definition and Documentation of Use Cases	Partial	60% (3/5)	Three use cases (Visualizza misure correnti, Richiedi aggiornamento specifico, Richiedi aggiornamento globale) are documented with descriptions, user goals (Livello: Obiettivo utente), actor, base flow, and alternative flows where applicable. The use case diagram shows relationships via «include» to Presenta elenco misure. However, the models do not specify explicit pre-conditions or post-conditions for the actions.
User interface and interaction design principles	Absent	29% (2/7)	The system uses a command-line interface and provides an activity diagram describing the interactive flow (menu, choices 1/2/3/0, calls to mostraMisurazioni and notificaMisura), which gives a clear navigation structure. However, there are no UI mockups, no reservation-related processes, no explicit validation mechanisms or structured input field descriptions, and no discussion of dynamic UI updates.
Separation of concerns in software architecture	Partial	40% (2/5)	The architecture section defines distinct packages domain, data, observer, and sensors, each with clearly described responsibilities, demonstrating separation of concerns and modular design. Interactions between these components are documented (e.g., domain depends on sensors and data; observer depends on data and java.util). However, there are no explicit MVC, Service, or DAO layers or packages named Model/View/Controller/Service, nor relationships between Controller, Service, and DAO layers.
Data Access Object (DAO) pattern	Absent	0% (0/7)	The document does not mention any DAO classes, DAO interfaces, CRUD operations, SQL queries, or database access abstraction. All data handling is in the Misurazione DTO and in-memory structures; there is no persistence layer or DAO pattern described or tested.

5 Summary Table

Category	HIGH	MEDIUM	LOW	Total
Architecture	0	0	1	1
Requirements	0	8	0	8
Testing	1	2	2	5
Total	**1**	**10**	**3**	**14**

6 Issue Details

6.1 Architecture (1 issues)

ISS-001 — LOW [100%] — Page 5 The architectural description states that domain handles Composite and Adapter and that observer contains the concrete Observer, but the document does not explicitly discuss how the CLI/controller layer depends on these packages. The activity diagram shows direct calls like "Chiama mostraMisurazioni()" and "Chiama notificaMisura() sull'adapter", but there is no class-level description of this controller. This makes the mapping between the designed architecture and the implemented interaction layer slightly opaque.

> • *domain. Definisce e implementa la struttura gerarchica dei componenti sensore tramite il pat- tern Composite (Component, Composite) e adatta i sensori specifici tramite il pattern Adapter (AdapterA/B/C), rendendoli anche osservabili (pattern Observer - Subject).*

Recommendation: Add a brief subsection in the architecture or implementation chapter describing the class (or main method) that implements the command-line interaction. Clarify in which package it resides, which classes it collaborates with (Centralina, Composite root, Adapter instances), and how it respects the separation between domain, observer, data, and sensors. A small class diagram or a short code snippet showing this controller would make the architecture-to-implementation mapping clearer and help reviewers understand where the use cases are realized.

6.2 Requirements (8 issues)

UC-1

ISS-009 — MEDIUM [90%] — Page 14 La sezione di collaudo descrive test a livello di unità per Adapter, Composite, Centralina, sensori e DTO, ma non esiste una mappatura esplicita tra i casi d'uso (Visualizza misure correnti, Richiedi aggiornamento specifico, Richiedi aggiornamento globale) e scenari di test che li coprano end-to-end. In particolare, non sono menzionati test che verifichino il flusso completo dalla scelta dell'utente nell'interfaccia a riga di comando fino alla visualizzazione delle misure, né test che coprano i flussi alternativi descritti nei casi d'uso (es. identificativo sensore non valido, assenza di misure). Questo rende difficile dimostrare che i requisiti funzionali e i casi d'uso siano effettivamente soddisfatti dal sistema nel suo insieme.

> *4 Collaudo unitario Sono stati implementati test unitarî utilizzando JUnit 5 per verificare la correttezza delle componenti chiave e l'implementazione dei pattern. ... 4.1 Test degli adapter I test unitarî per queste classi (es. AdapterATest) sono strutturalmente simili e mirano a verificare: • l'aderenza ai pattern: si controlla che l'adapter implementi l'interfaccia Component (per il pattern Composite) e che estenda la classe Observable (per il pattern Observer). Questo viene verificato tramite asserzioni instanceof; • il recupero della misura (ottieniMisura), verificando che il metodo deleghi correttamente la chiamata al sensore (simulato tramite un mock per isolare il test), restituisca il valore ottenuto in- capsulato nel formato richiesto dall'interfaccia Componente aggiorni lo stato interno dell'adapter (l'attributo ultimaMisuraValore); • la notifica all'observer (notificaMisura), che si assicura che la chiamata*

> *a notificaMisura recuperi un nuovo valore dal sensore (mock), crei un oggetto Misurazione contenente i dati corretti e lo invii agli observer registrati. ... 4.2 Test del Composite (CompositeTest) ... 4.3 Test dell'Observer concreto (CentralinaTest) ... 4.4 Test dei sensori (SensoreATest, ecc.) ... 4.5 Test della classe dati (MisurazioneTest)*

Recommendation: Aggiungi una sezione di tracciabilità requisiti–test o casi d'uso–test. Per ciascun caso d'uso (UC1, UC2, UC3), definisci almeno uno scenario di test di integrazione o di sistema che: (1) simuli l'interazione dell'utente tramite l'interfaccia a riga di comando (anche solo a livello di chiamate di metodi se non vuoi testare l'I/O reale); (2) verifichi che, dopo l'azione dell'utente, lo stato della Centralina (lista ultimeMisure) e l'output mostrato siano coerenti con le postcondizioni del caso d'uso; (3) copra i flussi alternativi principali (nessuna misura disponibile, identificativo sensore inesistente, errori di input). Anche se non implementi tutti questi test nel codice per motivi di tempo, descriverli nella relazione e collegarli ai requisiti migliorerà molto la qualità ingegneristica del documento.

General Issues

ISS-002 — MEDIUM [100%] — Page 3 I requisiti funzionali RF1–RF4 sono solo descrittivi e mancano di precondizioni, postcondizioni e criteri osservabili di successo/errore. Ad esempio non è specificato cosa succede se un sensore non risponde, se la centralina non è registrata come observer, o quali dati minimi debbano sempre essere presenti nella notifica. RF5 è un requisito di processo molto generico e non è collegato in modo tracciabile ai singoli casi d'uso o funzionalità da testare.

> *2.1 Requisiti funzionali Sono stati definiti i seguenti requisiti per guidare lo sviluppo del sistema. • RF1 (Acquisizione dati). Il sistema deve poter ottenere il valore misurato da ciascun tipo di sensore disponibile (tipo A, B, C), indipendentemente dalla sua interfaccia specifica. • RF2 (Adattamento obbligatorio). Le classi originali dei sensori (A, B, C) sono considerate componenti esterni non modificabili. Il sistema deve adattarle a un'interfaccia interna comune senza alterare le classi originali. • RF3 (Aggregazione sensori). Deve essere possibile creare gruppi di sensori. Il sistema deve permettere di richiedere tutte le misurazioni dei sensori appartenenti a un gruppo (o all'intera struttura) tramite un'unica operazione sull'elemento radice del gruppo. • RF4 (Notifica centralina). Il sistema deve includere una centralina che viene notificata auto- maticamente quando si forza l'aggiornamento dei sensori (a seguito di una richiesta di notifica). La notifica deve contenere informazioni identificative del sensore e il valore misurato. • RF5 (Collaudo unitario). Devono essere implementati test unitarî (JUnit) per verificare il corretto funzionamento.*

Recommendation: Per ciascun requisito funzionale RF1–RF4, aggiungi almeno: (1) precondizioni chiare (es. "tutti i sensori sono stati istanziati e registrati nel Composite", "almeno una Centralina è registrata come Observer"), (2) postcondizioni verificabili (es. "per ogni sensore gestito esiste una Misurazione aggiornata nella lista della Centralina"), (3) gestione degli errori (es. "se un sensore lancia un'eccezione, il sistema registra l'errore e continua con gli altri sensori"), e (4) eventuali vincoli sui dati (es. campi obbligatori nella Misurazione). Per RF5, sostituiscilo o integralo con un requisito di testabilità che leghi esplicitamente ogni requisito/caso d'uso a uno o più test (es. "Per RF3 devono esistere test che verifichino l'aggregazione di misure in presenza di più Adapter e Composite annidati").

ISS-003 — MEDIUM [100%] — Page 3 RF4 parla di "notifica" e di contenuto della notifica, ma non specifica in modo completo quali campi siano obbligatori nella Misurazione né come la centralina debba gestire notifiche non valide. L'implementazione di Centralina.update mostra che vengono gestiti solo argomenti di tipo Misurazione e che altri tipi vengono ignorati con un messaggio su console, ma questo comportamento non è descritto nei requisiti o nei casi d'uso. Manca quindi un allineamento tra requisito, modello di dati (Misurazione) e logica di business della Centralina.

> • *RF4 (Notifica centralina). Il sistema deve includere una centralina che viene notificata auto- maticamente quando si forza l'aggiornamento dei sensori (a seguito di una richiesta di notifica). La notifica deve contenere informazioni identificative del sensore e il valore misurato.*

Recommendation: Estendi RF4 per descrivere esplicitamente il contratto della notifica: (1) elenca i campi minimi che devono sempre essere presenti nella Misurazione (tipologia, nomeSensore, valore, unitaMisura, variazione) e specifica che le notifiche prive di questi dati sono considerate non valide; (2) descrivi cosa deve fare la Centralina quando riceve un argomento non di tipo Misurazione (ad esempio ignorarlo silenziosamente o registrare un errore), in modo coerente con il codice di update; (3) collega RF4 alla classe Misurazione (sezione 3.5) indicando che essa è il formato standard della notifica; (4) se necessario, aggiungi un requisito non funzionale sulla robustezza, ad esempio "Il sistema deve continuare a funzionare correttamente anche se riceve notifiche non valide, senza interrompere l'elaborazione delle notifiche successive".

ISS-004 — MEDIUM [92%] — Page 4 Il caso d'uso "Richiedi aggiornamento specifico" non specifica come l'"identificativo del sensore" venga mappato sugli Adapter/Component nel Composite, né cosa succede se il sensore esiste ma non ha ancora mai prodotto misure. Non è descritto se l'aggiornamento avviene tramite chiamata diretta al sensore o tramite notificaMisura() dell'Adapter (che a sua volta notifica la Centralina). Inoltre, il flusso alternativo copre solo il caso "sensore inesistente" ma non gestisce input non valido (stringa vuota, formato errato) o errori di lettura dal sensore.

> *Caso d'uso 2 Richiedi aggiornamento specifico Descrizione L'utente richiede al sistema di ottenere una nuova misura da un sensore specifico e di visualizzare lo stato aggior- nato. ... Flusso base 1. L'utente seleziona l'opzione per aggiornare un sensore specifico. 2. Il sistema chiede all'utente di specificare quale sensore aggiornare. 3. L'utente fornisce l'identificativo del sensore. 4. Il sistema richiede e ottiene una nuova lettura dal sen- sore specificato. 5. Il sistema aggiorna l'ultima misura nota per quel sen- sore. 6. Il sistema presenta all'utente l'elenco aggiornato delle ultime misure disponibili per tutti i sensori. Flusso alternativo Se l'identificativo fornito dall'utente non corrisponde a un sensore gestito dal sistema, il sistema informa l'utente dell'errore.*

Recommendation: Nel caso d'uso 2, specifica: (1) il formato dell'identificativo (es. nome del sensore, indice, codice univoco) e come viene risolto in un oggetto Adapter/Component; (2) che l'aggiornamento avviene invocando notificaMisura() sull'Adapter corrispondente, che a sua volta legge dal sensore e notifica la Centralina; (3) una postcondizione chiara: "Se la lettura ha successo, la lista ultimeMisure della Centralina contiene una Misurazione aggiornata per quel sensore"; (4) flussi alternativi aggiuntivi per input non valido (identificativo vuoto o malformato) e per fallimento della lettura (es. eccezione dal sensore), specificando se il sistema mostra un messaggio di errore e mantiene la misura precedente in Centralina.

ISS-005 — MEDIUM [93%] — Page 4 Nel caso d'uso "Richiedi aggiornamento globale" non è definito il comportamento in presenza di errori parziali: cosa succede se uno o più sensori falliscono la lettura mentre altri rispondono correttamente? Non è specificato se l'operazione debba essere "tutto o niente" o se si accettano aggiornamenti parziali. Inoltre, non è chiarito se l'aggiornamento globale sia implementato tramite una singola chiamata notificaMisura() sul Composite (che delega ai figli) o tramite iterazioni esplicite sugli Adapter, e come questo si rifletta sulla Centralina.

> *Caso d'uso 3 Richiedi aggiornamento globale Descrizione L'utente richiede al sistema di ottenere una nuova misura da tutti i sensori gestiti e di visualizzare lo stato comples- sivo aggiornato. ... Flusso base 1. L'utente seleziona l'opzione per aggiornare tutti i sen- sori. 2. Il sistema richiede e ottiene una nuova lettura da cia- scun sensore gestito. 3. Il sistema*

> *aggiorna l'ultima misura nota per ogni sen- sore. 4. Il sistema presenta all'utente l'elenco aggiornato delle ultime misure disponibili per tutti i sensori.*

Recommendation: Arricchisci il caso d'uso 3 specificando: (1) che l'aggiornamento globale è realizzato invocando notificaMisura() sull'oggetto Composite radice, che a sua volta chiama notificaMisura() su tutti i figli (come descritto in Listato 3); (2) una politica chiara per gli errori parziali, ad esempio "Il sistema tenta di aggiornare tutti i sensori; per quelli che falliscono, mantiene la misura precedente e segnala all'utente quali sensori non sono stati aggiornati"; (3) una postcondizione che distingua tra sensori aggiornati con successo e sensori rimasti con misure vecchie; (4) eventuali vincoli temporali o di ordine (es. se l'utente può richiedere un nuovo aggiornamento globale mentre uno è ancora in corso, anche solo concettualmente).

ISS-006 — MEDIUM [93%] — Page 4 Nel caso d'uso "Visualizza misure correnti" non è specificato da dove provengano le "ultime misure" né come si relazionino con la logica Observer descritta in seguito. Non è chiaro se la visualizzazione legge direttamente dai sensori, dal Composite, o dalla lista interne della Centralina aggiornata via Observer. Inoltre, non è definito cosa succede se alcuni sensori hanno misure e altri no, o se la centralina non è stata ancora registrata come observer: il flusso alternativo tratta solo il caso "nessuna misura in assoluto".

> *Caso d'uso 1 Visualizza misure correnti Descrizione L'utente richiede di vedere l'ultimo stato noto delle mi- sure dei sensori gestiti dal sistema. ... Flusso base 1. L'utente seleziona l'opzione per visualizzare le misure correnti. 2. Il sistema presenta all'utente l'elenco delle ultime mi- sure disponibili per ciascun sensore conosciuto. Flusso alternativo Se il sistema non dispone ancora di alcuna misura, infor- ma l'utente che non ci sono dati disponibili da visualiz- are.*

Recommendation: Esplicita nel caso d'uso 1 che la fonte dei dati è la Centralina (lista ultimeMisure) e non una lettura diretta dai sensori, allineandoti con l'implementazione di Centralina.mostraMisurazioni(). Aggiungi: (1) una precondizione del tipo "La Centralina è stata registrata come Observer di tutti gli Adapter"; (2) una postcondizione che descriva che nessuno stato viene modificato, solo letto; (3) flussi alternativi per casi parziali, ad esempio "Se esistono misure solo per alcuni sensori, il sistema mostra solo quelle disponibili e indica chiaramente quali sensori non hanno ancora misure"; (4) un collegamento esplicito al metodo mostraMisurazioni() per coerenza con il diagramma di attività (figura 4).

ISS-007 — MEDIUM [100%] — Page 4 Tutti i casi d'uso sono descritti solo con un flusso base e, in due casi, un singolo flusso alternativo. Mancano sistematicamente precondizioni (es. esistenza di sensori, centralina inizializzata, observer registrati), postcondizioni (stato atteso della lista di misure nella centralina, stato dei sensori dopo l'aggiornamento) e scenari di errore più articolati (es. fallimento di un singolo sensore durante l'aggiornamento globale, input non numerico o vuoto per l'identificativo del sensore). Questo rende difficile per il lettore e per chi testa capire esattamente quali stati del sistema sono garantiti dopo ogni interazione.

> *2.2 Casi d'uso I casi d'uso descrivono le interazioni principali tra l'utente e il sistema. L'applicativo ha un solo attore, l'utente, che s'interfaccia con la centralina. L'utente ha la possibilità di visualizzare le misure arrivate alla centralina, di richiedere l'aggiornamento di un sensore specifico o di richiedere l'aggiornamento di tutti i sensori. Gli schemi qui riportati rappresentano i casi d'uso.*

Recommendation: Per tutti i casi d'uso (Visualizza misure correnti, Richiedi aggiornamento specifico, Richiedi aggiornamento globale), aggiungi esplicitamente: (1) Precondizioni, ad esempio "Almeno un sensore è stato creato e registrato nel Composite" e "La Centralina è registrata come Observer di tutti gli Adapter"; (2) Postcondizioni, ad esempio "La lista interne

ultimeMisure della Centralina contiene al più una Misurazione per ogni sensore" o "Per il sensore aggiornato è stata sostituita la Misurazione precedente"; (3) Flussi alternativi per errori tecnici (sensore non raggiungibile, eccezioni, nessun sensore registrato) e per input non valido (identificativo vuoto, formato errato). Usa una struttura tabellare o numerata per distinguere chiaramente flusso principale, alternative e condizioni di ingresso/uscita per ogni UC.

ISS-008 — MEDIUM [91%] — Page 7 Il diagramma di sequenza mostra l'Utente che interagisce con la Centralina, la quale chiama ottieniMisura() sul Composite per ottenere una lista di valori e poi "Mostra lista misure". Tuttavia, nei casi d'uso e nella descrizione dell'Observer, la visualizzazione delle misure è legata alla lista interne della Centralina aggiornata tramite notifiche (Misurazione), non a una lista di Double restituita da ottieniMisura(). Questa doppia modalità (lettura pull via Composite vs. push via Observer) non è chiarita nei requisiti o nei casi d'uso, creando una potenziale incoerenza concettuale su quale sia il flusso principale di business.

> *2.7 Diagramma di sequenza: ottenere tutte le misure Illustra come la richiesta di misure viene gestita dalla struttura Composite. [IMAGE 1]: Diagram Type: Sequence Diagram Elements: - Utente - Centralina - Composite - AdapterA - A - AdapterB - B - Composite - AdapterC - C Relationships: - Utente interacts with Centralina. - Centralina sends messages to Composite: 'ottieniMisura()'. ... - List [valoreA, valoreB, valoreC] - Mostra lista misure*

Recommendation: Decidi e documenta chiaramente quale sia il flusso principale per "ottenere tutte le misure": (1) Se il flusso principale è basato su Observer (Adapter notifica Centralina con Misurazione e la Centralina mostra le sue ultimeMisure), allora aggiorna il diagramma di sequenza per mostrare notificaMisura() sugli Adapter e update() sulla Centralina, e fai sì che i casi d'uso facciano riferimento a questo flusso; (2) Se invece vuoi mantenere anche un flusso pull tramite ottieniMisura(), esplicita nei requisiti e nei casi d'uso quando si usa l'uno o l'altro (ad esempio: "Visualizza misure correnti" usa solo la lista della Centralina, mentre un caso d'uso tecnico potrebbe usare ottieniMisura() per diagnosi). In entrambi i casi, assicurati che la descrizione testuale dei casi d'uso e la documentazione dell'Observer siano coerenti con il diagramma di sequenza e con il diagramma di attività (figura 4).

6.3 Testing (5 issues)

TST-001 — HIGH [100%] — Page 3 There is no explicit traceability from RF1 and the use cases to the described unit tests: tests are defined per class (Adapter, Composite, Centralina, sensors, Misurazione) but the document does not show any test that validates the complete functional flows of RF1/RF3 (e.g., from user request through Composite to all adapters and back to the centralina output). This makes it hard to demonstrate that the main acquisition and aggregation requirements are actually verified end-to-end.

> • *RF1 (Acquisizione dati). Il sistema deve poter ottenere il valore misurato da ciascun tipo di sensore disponibile (tipo A, B, C), indipendentemente dalla sua interfaccia specifica.*

Recommendation: Add a short subsection or table that maps each functional requirement and each use case to one or more concrete test classes/methods. For example, specify that RF1 and RF3 / "Richiedi aggiornamento globale" are covered by methods like CompositeTest.testOttieniMisuraAggregaTuttiISensori() and CentralinaTest.testMostraMisurazioniDopoAggiornamentoGlobale(). If such tests do not yet exist, add at least one integration-style JUnit test that: (1) builds a small Composite with AdapterA/B/C, (2) triggers ottieniMisura() or notificaMisura() as in the sequence/activity diagrams, and (3) asserts that the resulting list of values and/or the centralina's stored Misurazione objects match expectations. Then reference these tests explicitly under RF1/RF3 to make traceability clear for the examiner.

TST-002 — MEDIUM [100%] — Page 4 Alternative flows and error conditions described in the use cases (no measurements available, invalid sensor identifier, invalid menu choice) are not mentioned in the unit test descriptions. Tests focus on happy paths and internal class behavior, but the document does not indicate any tests that verify these negative/alternative scenarios, so these behaviors are not demonstrably covered.

> *Flusso alternativo Se il sistema non dispone ancora di alcuna misura, infor- ma l'utente che non ci sono dati disponibili da visualiz- zare.*

Recommendation: For each use case with an alternative or error flow, add at least one dedicated test method. Examples: (1) For "Visualizza misure correnti" add a test in CentralinaTest that initializes a Centralina with an empty ultimeMisure list, calls mostraMisurazioni(), and asserts that the console output contains the message equivalent to "non ci sono dati disponibili". (2) For "Richiedi aggiornamento specifico" add a test around the CLI controller (or a thin façade) that passes an unknown sensor ID and verifies that the system prints the error message described in the use case. (3) For the activity diagram's "Altro" branch, add a test that simulates an invalid menu choice and checks that the error message is produced and that the loop continues. Document these tests explicitly as covering the alternative flows so the mapping is clear.

TST-003 — MEDIUM [100%] — Page 7 The tests described for Adapter, Composite, Centralina, and sensors operate at class level and do not exercise the command-line interaction logic shown in the activity diagram (menu, reading user choice, mapping to operations, handling invalid IDs). As a result, the behavior of the interactive layer that connects user input to domain operations and error messages is not covered by tests.

> *Diagramma di attività: flusso interattivo Descrive il flusso logico dell'interfaccia utente a riga di comando. [IMAGE 2]: Diagram Type: Flowchart Elements: - Setup iniziale (crea oggetti) - Mostra menu - Leggi scelta utente - Scelta? - Mostra misurazioni attuali - Chiama mostraMisurazioni() - Aggiorna sensore specifico - Chiama notificaMisura() sull'adapter - Forza aggiornamento sensori - Chiama notificaMisura() - Imposta uscita - Messaggio errore - Uscita? - no - Chiudi centralina*

Recommendation: Introduce a thin controller or CLI handler class (if not already present) and write a small set of JUnit tests around it. Use dependency injection or simple test doubles for Centralina and the Composite root so that tests can: (1) simulate user choices "1", "2" with a valid and an invalid sensor ID, "3", "0", and an invalid option; (2) verify that the correct domain methods (mostraMisurazioni, notificaMisura on the right adapter or on the root Composite) are invoked; and (3) assert that the expected messages (including error messages for invalid IDs and invalid menu options) are printed. Then, in the document, explicitly state that these tests cover the interactive flows of the three use cases and their alternatives.

TST-004 — LOW [100%] — Page 14 Pattern conformance for Adapter classes is only checked via instanceof assertions, which confirms type relationships but not the behavioral contracts implied by the patterns (e.g., that Composite can treat adapters uniformly as Component, or that observers are actually notified when expected). This limits the strength of the tests in demonstrating correct pattern usage.

> *I test unitarî per queste classi (es. AdapterATest) sono strutturalmente simili e mirano a verificare: • l'aderenza ai pattern: si controlla che l'adapter implementi l'interfaccia Component (per il pattern Composite) e che estenda la classe Observable (per il pattern Observer). Questo viene verificato tramite asserzioni instanceof;*

Recommendation: Keep the instanceof checks, but complement them with behavioral tests that exercise the patterns through their common interfaces. For example: (1) In CompositeTest, build a Composite that contains real AdapterA/B/C instances (with mocked sensors), call ottieniMisura() and notificaMisura() only through the Component interface, and assert that the aggregated results and notifications are correct. (2) In CentralinaTest, register Centralina directly on an Adapter instance and trigger notificaMisura() via the Component reference, verifying that update() is called and ultimeMisure is updated. Document these tests as evidence that the patterns are not only structurally but also behaviorally respected.

TST-005 — LOW [100%] — Page 15 CentralinaTest verifies mostraMisurazioni by capturing console output, but the document does not indicate any tests that combine Centralina with real Adapter instances and real notifications in a single scenario. Tests are split (Adapter tests with TestObserver, Centralina tests with TestObservable), so there is no explicit test that the whole Observer chain (Adapter -> Centralina -> mostraMisurazioni) works together as in the use cases.

> • *che il metodo mostraMisurazioni rifletta accuratamente lo stato interno corrente della centra- lina (verificato tramite cattura dell'output su console).*

Recommendation: Add at least one integration-style JUnit test that wires together a real Adapter (with a mock sensor), a real Centralina, and the Observer registration. The test should: (1) register Centralina as observer on the Adapter, (2) set a known value on the mock sensor, (3) call adapter.notificaMisura(), and then (4) call centralina.mostraMisurazioni() and assert (via captured output or by inspecting ultimeMisure if accessible) that the printed measurement matches the expected Misurazione. Document this test as demonstrating the complete Observer flow used in the use cases.

7 Priority Recommendations

The following actions are considered priority:

1. **TST-001** (p. 3): Add a short subsection or table that maps each functional requirement and each use case to one or more concrete test classes/methods.

8 Traceability Matrix

Of 8 traced use cases: 6 fully covered, 0 without design, 2 without test.

ID	Use Case	Design	Test	Gap
RF1	Acquisizione dati	✓	✓	—
RF2	Adattamento obbligatorio	✓	✓	—
RF3	Aggregazione sensori	✓	✓	—
RF4	Notifica centralina	✓	✓	—
RF5	Collaudo unitario	✓	✓	—
UC1	Visualizza misure correnti	✓	✓	—
UC2	Richiedi aggiornamento specifico	✓	×	There is no explicit test that drives the full interactive flow for a single-sensor update (selection of a specific adap...

ID	Use Case	Design	Test	Gap
UC3	Richiedi aggiornamento globale	✓	×	There is no end-to-end test that exercises the global update scenario via the Composite root (calling notificaMisura() o...

9 Terminological Consistency

Found **10** terminological inconsistencies (0 major, 10 minor).

Group	Variants found	Severity	Suggestion
Command-line interface naming	"interfaccia a riga di comando", "interfaccia utente a riga di comando", "flusso interattivo", "flusso interattivo principale"	MINOR	Use a single, consistent term for the command-line interface, e.g., always "interfaccia a riga di comando" when describing the UI layer.
Unit test naming	"Collaudo unitario", "collaudi unitarî", "test unitarî"	MINOR	Use one term consistently for the unit tests section, e.g., always "test unitarî" in Italian text.
DTO naming	"Data transfer object", "data transfer object", "DTO"	MINOR	Use a single term for the DTO concept, e.g., always "data transfer object" plus the acronym in first occurrence ("data transfer object (DTO)") and then only "DTO".
Observer role naming	"Observer", "osservatore", "osservatori", "Observer concreto"	MINOR	Use one consistent label for the observer role, e.g., always "Observer" when referring to the pattern role, and reserve "osservatore" only for explanatory Italian text if needed.
Observable / subject naming	"Osservabile (soggetto)", "soggetto", "osservabile"	MINOR	Use a single term for the observable/subject role, e.g., always "Observable" for the Java type and "soggetto" or "osservabile" consistently when describing the role in Italian.
Composite client naming	"cliente", "client", "applicativo"	MINOR	Use one consistent term for the client of the Composite, e.g., always "cliente" or always "client" depending on the chosen language style.
Sensor class vs physical sensor naming	"classi di sensori", "classi (A, B, C)", "sensori fisici originali"	MINOR	Use a single term for the sensor classes, e.g., always "classi dei sensori" when referring to A, B, C, and reserve "sensori fisici originali" only when explicitly contrasting with simulation.

Group	Variants found	Severity	Suggestion
Last measurement value naming	"ultimaMisuraValore", "ultima misura nota", "ultima misura"	MINOR	Use a single term for the internal state representing the last value, e.g., always "ultimaMisuraValore" in prose when referring to that attribute.
Measurement list naming	"elenco delle ultime misure", "elenco misure", "lista misure"	MINOR	Use one consistent term for the list of measurements, e.g., always "elenco delle ultime misure" when referring to what is shown to the user.
System / program naming	"sistema", "programma", "applicativo"	MINOR	Use a single term for the overall software artifact, e.g., always "applicativo" or always "programma".