

UNIVERSITÀ DEGLI STUDI DI SALERNO
DIPARTIMENTO DI INFORMATICA

Ingegneria del Software Tecniche Avanzate

UniClass - Maintenance Report
Preventive Re-engineering and Architectural Refactoring

Team

Lucageneroso Cammarota (Matricola: NF22500053)
Giuseppe Sabetta (Matricola: NF22500155)



ANNO ACCADEMICO 2025/2026

Indice

1	Introduzione	4
2	Baseline Assessment	5
2.1	Analisi di Portafoglio e Posizionamento Strategico	6
2.2	Metriche Quantitative della Baseline - Analisi Statica	6
3	Reason for Change	8
4	Riferimenti e acronimi	10
5	Change Decision & Impact Analysis	12
5.1	Decisione Strategica: Re-engineering vs. Maintenance	12
5.2	Risk Evaluation e Classificazione	12
5.3	Metodologia di Impact Analysis	13
5.3.1	Costruzione del Candidate Impact Set (CIS)	13
5.3.2	Dependencies Graph	13
5.3.3	Matrice di connettività intramodulo (Utenti)	13
5.3.4	Matrice di Connettività Intermodulare (Baseline)	14
5.3.5	Quantificazione del Ripple Effect	15
5.3.6	Matrice di Raggiungibilità (Baseline)	15

5.4	Tracciabilità (Traceability)	16
5.5	Organizzazione e Pianificazione	17
6	Reengineering Pipeline	18
7	Reverse Engineering	19
7.1	Costruzione del Candidate Impact Set (CIS)	19
7.2	Analisi Quantitativa tramite Matrici di Raggiungibilità	19
8	Alteration	21
8.1	Redesign Architetturale: Composition over Inheritance	21
8.2	Migrazione del Modello Dati (ETL)	22
9	Forward Engineering	24
9.1	Implementazione del Codice	24
9.2	Refactoring della Test Suite	24
10	Validation & Verification	25
10.1	Risultati della Baseline (Pre-Refactoring) - Funzionale	25
10.1.1	Sintesi Quantitativa	25
10.1.2	Analisi della Copertura per Modulo	25
10.2	Risultati della Baseline (Pre-Refactoring) - Non Funzionale	26
10.2.1	MicroBenchmark Metrics	26
10.2.2	System Performance Testing	26
10.3	Differenze Test Execution e Copertura	27
10.3.1	Testing Funzionale	27
10.3.2	Testing Non Funzionale	27

10.4 Mutation Testing	27
11 Risultati	29
11.1 Sintesi dei risultati	29
11.1.1 Analisi Statica	29
11.1.2 Matrice di Connettività (Target - Post Refactoring)	29
11.1.3 Matrice di Raggiungibilità (Target - Post Refactoring)	30
11.1.4 Ripple Effects: Baseline Vs Target	31
11.1.5 Definizione delle Metriche	31
11.1.6 Analisi Comparativa	32
11.1.7 Interpretazione dei Risultati	32
11.2 Comparazione con la baseline	33
11.2.1 Analisi delle dipendenze del modulo riingegnerizzato	33
11.3 Confronto tra le dipendenze del modulo <i>utenti</i> : baseline vs. target post-refactoring .	34
11.3.1 Analisi Comparativa	35
11.3.2 Analisi delle dipendenze delle componenti	36
12 Conclusioni	37

Capitolo 1

Introduzione

Il presente documento costituisce il rapporto tecnico finale relativo all'intervento di manutenzione preventiva ed evolutiva effettuato sul sistema *UniClass*, una piattaforma di gestione universitaria progettata per l'amministrazione di utenti, risorse didattiche e orari. L'evoluzione del sistema è stata guidata da una *Change Request (CR)* finalizzata al superamento dei limiti strutturali del modello legacy, basato su una rigida gerarchia di ereditarietà (*Is-A*), e alla transizione verso un'architettura più flessibile, fondata su composizione e ruoli dinamici.

L'intervento, inquadrato secondo lo standard ISO/IEC 14764:2006 e le linee guida del SWEBOK Guide v4.0, ha seguito un processo sistematico di *Reengineering*. Questo ha compreso una fase di analisi del portafoglio software (*Portfolio Analysis*), un'approfondita *Impact Analysis* per la gestione del rischio di propagazione (*Ripple Effect*), e un rigoroso ciclo di validazione tramite test di regressione e *Mutation Testing*.

L'obiettivo primario è stato la riduzione del debito tecnico accumulato, l'incremento della manutenibilità e il supporto a requisiti funzionali complessi, quali la gestione multi-ruolo degli utenti accademici, preservando al contempo l'integrità funzionale del sistema.

Capitolo 2

Baseline Assessment

La fase preliminare di valutazione (*Baseline Assessment*) ha richiesto un'analisi quantitativa dello stato di fatto (*As-Is*) per giustificare razionalmente l'intervento. L'analisi si è avvalsa di strumenti di analisi statica (SonarCloud) e della costruzione di grafi di dipendenza per misurare il degrado strutturale.

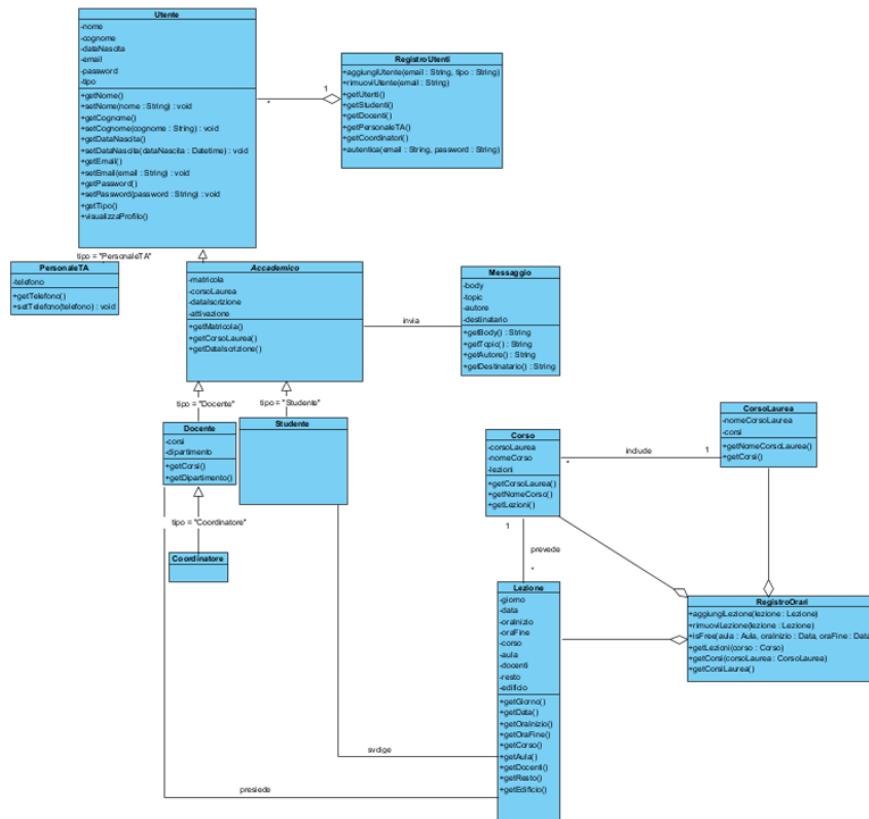


Figura 2.1: Mapping a oggetti pre alteration

2.1 Analisi di Portafoglio e Posizionamento Strategico

Applicando il modello decisionale *KTIM* (Keep, Tolerate, Invest, Migrate) descritto nella *Portfolio Analysis*, il modulo Utenti è stato posizionato nel quadrante critico **High Value / High Debt**. Come illustrato nella Figura ??, il modulo presenta un *Business Value* massimo (essendo hub centrale per l'autenticazione) ma un *Technical Debt* insostenibile.

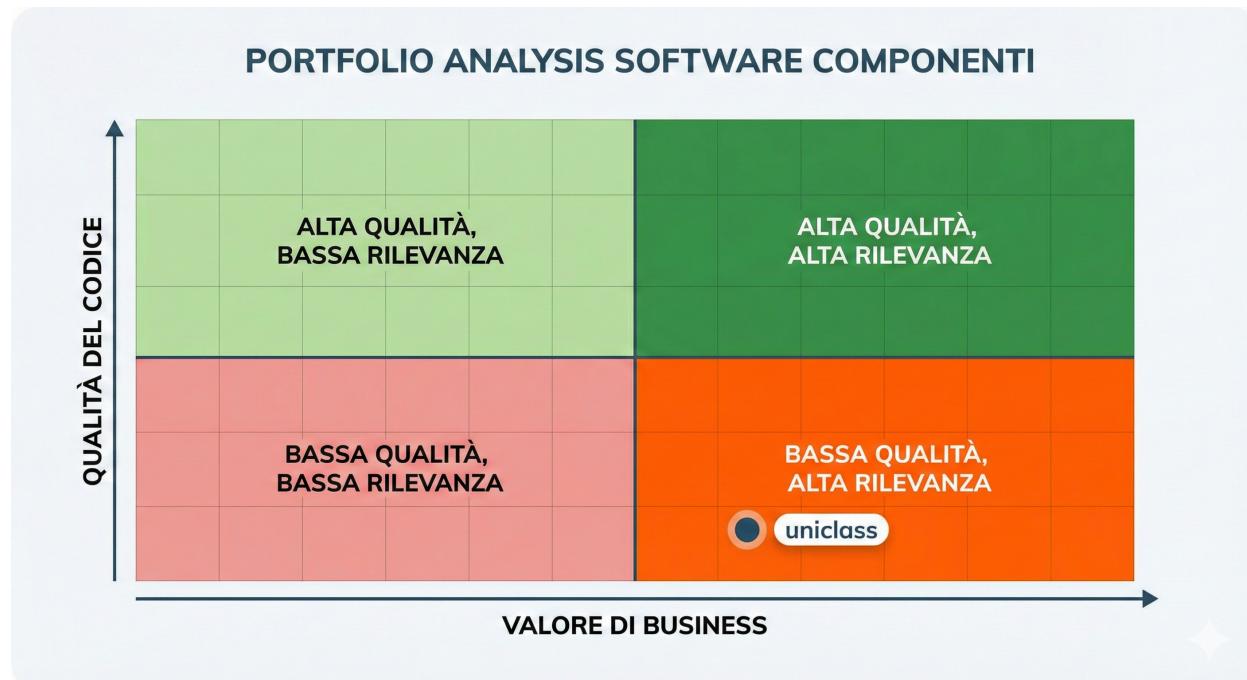


Figura 2.2: Portfolio Analysis Modulo Utenti

2.2 Metriche Quantitative della Baseline - Analisi Statica

Per completare l'analisi del punto di partenza, è stata condotta un'analisi con SonarCloud, impostando come quality profile il default-built in SonarWay. Degno di nota, da questo punto di vista, il valore relativo alla Maintainability che guadagna grado A.

L'analisi statica ha evidenziato metriche di complessità fortemente fuori soglia rispetto agli standard di riferimento (SWEBOK, Cap. 15). I dati sono riassunti nella Tabella 2.1.

Categoria	Metrica	Valore	Interpretazione
Complessità	Cyclomatic Complexity	210	Concentrazione del 35.9% della complessità totale del sistema.
Complessità	Cognitive Complexity	97	Bassa leggibilità e alta difficoltà di manutenzione.
Accoppiamento	Fan-In Transitivo	15+	<i>Hotspot</i> logico primario (God Object candidate).
Accoppiamento	Cross-Module Coupling	5	Violazione dell'incapsulamento tra moduli.

Tabella 2.1: Metriche statiche estratte dalla Baseline (SonarCloud).

L'elevata complessità ciclomatica e cognitiva conferma la presenza di un *God Service* nel layer applicativo, rendendo il sistema fragile rispetto a modifiche evolutive e incrementando esponenzialmente i costi di manutenzione.

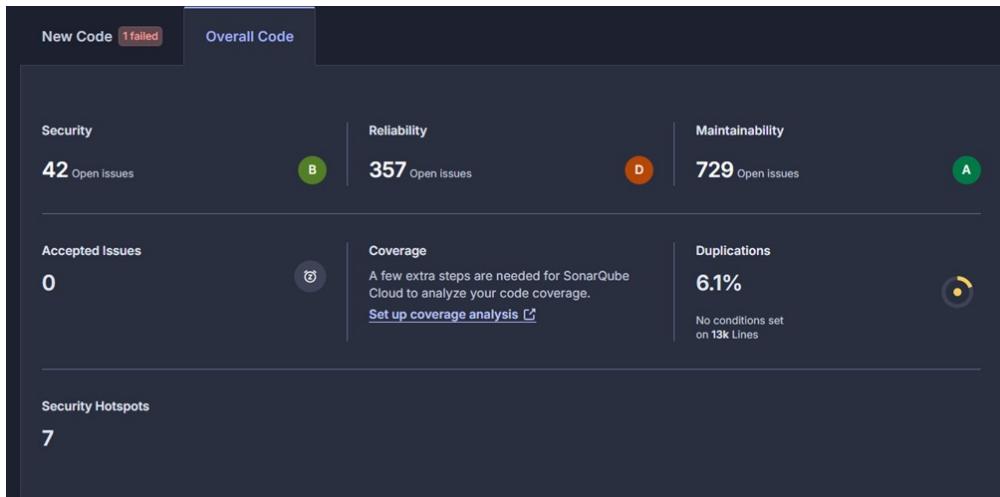


Figura 2.3: Dashboard SonarCloud dopo l'analisi della baseline

Capitolo 3

Reason for Change

La motivazione al cambiamento non deriva da un mero bisogno estetico di ristrutturazione, ma da impellenti limiti strutturali e funzionali che compromettono l'evoluzione del sistema. Le ragioni sono state categorizzate e correlate ai dati quantitativi nella Tabella 3.1.

Problema	Dato Quantitativo/Diagnostico	Impatto Funzionale/Operativo
Rigidità del Dominio	Modello "Is-A" rigido; Classe Accademico non astratta ma istanziata.	Impossibilità di gestire profili ibridi (es. Dottorando che è sia Studente che Docente). Necessità di duplicazione account.
Persistenza Inefficiente	Strategia <i>Joined Table</i> ; query log analysis mostra overhead iniziale (18-20ms).	Alto numero di JOIN per ricostruire oggetti complessi; difficoltà nel mantenimento dell'integrità referenziale.
Accoppiamento Forte	Fan-In Transitivo = 15; XMC = 5 violazioni cross-module.	<i>Ripple Effect</i> incontrollato: una modifica a AccademicoService impatta Controller, DAO e JSP correlate.
God Object/Service	Cyclomatic Complexity 210 (Service Layer = 107).	Logica di business accentuata e dispersiva; violazione del <i>Single Responsibility Principle</i> .

Tabella 3.1: Analisi delle ragioni del cambiamento e correlazione ai dati diagnostici.

L'analisi dei *Call Graph* (Allegato A.1 della CR) ha evidenziato che la separazione dei ruoli tramite ereditarietà non guida i flussi di esecuzione a runtime, ma introduce solo complessità struttura-

le. Ne consegue che il modello attuale rappresenta un ostacolo alla scalabilità e all'estensibilità, giustificando un intervento di *Reengineering Architetturale*.

Capitolo 4

Riferimenti e acronimi

Riferimenti Normativi

- ISO/IEC 14764:2006: Software Engineering — Software Life Cycle Processes — Maintenance.
- ISO/IEC/IEEE 29119: Software and systems engineering — Software testing.
- SWEBOK Guide v4.0: Guide to the Software Engineering Body of Knowledge.

Riferimenti e Navigazione della Documentazione

Per migliorare la coesione, understandability e coerenza degli artefatti ad alto livello d'astrazione, si è deciso di migliorare la documentazione. Si è notato, durante la riscrittura e durante il processo di Reengineering, che le configurazioni e diversi modelli rappresentativi lo scambio di messaggi all'interno delle funzionalità della piattaforma erano mancanti in parte o nella loro totalità. Si determina, di conseguenza, una lista di documenti con relativo contenuto e modifiche effettuate.

- **Requirements Analysis Document:** Si è revisionata la totalità dei Sequence Diagram e Casi d'uso, convergendo in una modifica e aggiunta di quest'ultimi per aumentare la coerenza del documento.
- **System Design Document:** Sono state modificate le sezioni di Decomposizione in sottosistemi, gestione dati persistenti e i servizi.
- **Object Design Document:** Si sono modificate le sezioni riguardanti il Mapping a Oggetti, con relativo modello, i packages presenti e le interfacce utilizzate per le funzionalità descritte in OCL della piattaforma.
- **Test Plan:** Data la presenza del TPTCS, ovvero Test Plan and Test Case Specifications, si è separata e distinta la responsabilità tra questi documenti uniti erroneamente in precedenza,

seguendo lo standard IEEE 829 alla lettera per tipologia, livello e metodologia del Testing presente nella baseline originaria e post-refactoring del sistema. Si definiscono quindi le tipologie funzionali e non funzionali del sistema, garantendo la possibilità di visionare le differenze prestazionali unitarie e sistemiche del sistema sulla base delle modifiche pianificate.

- **Test Case Specifications:** Sono quindi presenti i Test Case Specifications, Test Frames e Test Cases designati e implementati ulteriormente revisionati a partire dalla baseline originaria.
- **Portfolio Analysis** - Analisi Pre-Refactoring e Post-Refactoring del Technical Debt e Valore di Business del sistema UniClass con inquadramento metodologico e valutazione e validazione delle metriche ottenute.
- **Test Execution Report:** Nel documento è possibile (ed è consigliato) visionare i risultati funzionali e non funzionali dei test d'unità e di sistema, valutando le motivazioni e le metriche presenti nella differenza Pre-refactoring e Post-refactoring con discussione riguardo l'output ottenuto.

Inoltre, ogni documento originario, se non creato da zero, è stato aggiornato in LaTeX attraverso automatismi e interventi umani longevi e precisi, garantendo reperibilità delle informazioni e maggiore modularità della documentazione.

Acronimi

- **CIS:** Candidate Impact Set
- **CR:** Change Request
- **DAO:** Data Access Object
- **ETL:** Extract, Transform, Load
- **JPA:** Jakarta Persistence API
- **KTIM:** Keep, Tolerate, Invest, Migrate
- **NPE:** Null Pointer Exception
- **PIT:** Practical Inference Testing (Mutation Tool)
- **RTS:** Regression Test Selection
- **SIS:** Starting Impact Set
- **TER:** Test Execution Report
- **XMC:** Cross-Module Coupling

Capitolo 5

Change Decision & Impact Analysis

La fase decisionale rappresenta il cardine metodologico tra l'identificazione del problema e l'esecuzione dell'intervento. In accordo con lo standard ISO/IEC 14764, la scelta della strategia di manutenzione non è stata lasciata all'intuizione, ma supportata da un'analisi quantitativa del rischio e della portata dell'impatto.

5.1 Decisione Strategica: Re-engineering vs. Maintenance

Dalla *Portfolio Analysis* (Cap. 2), il modulo Utenti è emerso come un asset critico (*High Value*) ma degradato (*High Debt*). Secondo il modello decisionale KTIM e le teorie di Harry Sneed, per i componenti in questo quadrante, la manutenzione correttiva o adattiva è economicamente svantaggiosa: il debito tecnico incrementerebbe i costi futuri in modo esponenziale. Di conseguenza, è stata presa la decisione strategica di intraprendere un processo di **Reengineering Architetture Preventivo**, mirato a collassare la gerarchia rigida a favore di un modello a composizione, traslando il componente verso il quadrante *High Value / Low Debt*.

5.2 Risk Evaluation e Classificazione

Il documento di Change Request ha classificato il rischio dell'intervento come **ALTO**. Questa classificazione deriva dall'analisi metrica della baseline:

- **Complessità Ciclomatica (210)**: L'elevato numero di percorsi indipendenti rende difficile garantire una *Recall* del 100% nell'analisi di impatto, aumentando il rischio di *False Negatives* (bug che sfuggono all'analisi).
- **Complessità Cognitiva (219)**: Logica fortemente annidata che aumenta la probabilità di effetti collaterali (*Side Effects*) durante il refactoring.

- **Change Propagation:** L'analisi ha evidenziato un alto potenziale di propagazione delle modifiche dai DAO fino al layer di presentazione (JSP).

5.3 Metodologia di Impact Analysis

Per gestire il rischio alto, è stata condotta un'Impact Analysis formale, supportata da strumenti di Reverse Engineering (IntelliJ IDEA Dependency Analysis) e algoritmi di teoria dei grafi.

5.3.1 Costruzione del Candidate Impact Set (CIS)

L'analisi ha permesso di transitare dallo *Starting Impact Set (SIS)*, definito intuitivamente nella CR, al *Candidate Impact Set (CIS)*, calcolato rigorosamente. Il processo ha previsto:

1. **Estrazione Dipendenze:** Costruzione del grafo $G = (V, E)$ dove V sono le classi e E le relazioni di chiamata.
2. **Calcolo Matrici:** Generazione della Matrice di Adiacenza A e della Matrice di Raggiungibilità R (tramite algoritmo di Warshall).

La cardinalità finale del CIS è stata quantificata in **88 elementi**, classificati in Primary Impact Set (componenti direttamente modificate) e Secondary Impact Set (componenti affette da Ripple Effect).

5.3.2 Dependencies Graph

Utilizzando il tool fornito da IntelliJ, sono state analizzate le backward e forward dependencies delle componenti direttamente coinvolte nello SIS e di quelle strettamente correlate ad esse. Così facendo, si è identificato in AccademicoService una God Class con svariate dipendenze.

5.3.3 Matrice di connettività intramodulo (Utenti)

Per motivi di leggibilità, si riportano le matrici estratte limitatamente al sottosistema critico oggetto di refactoring (*Modulo Utenti*). Le matrici complete sono disponibili in allegato.

Baseline

Nella Tabella 5.1, l'elevata densità della colonna relativa ad AccademicoService evidenzia un accoppiamento forte (*High Fan-In*). I servizi specifici (StudenteService, DocenteService) mostrano dipendenza diretta sia dal servizio base che dal modello.

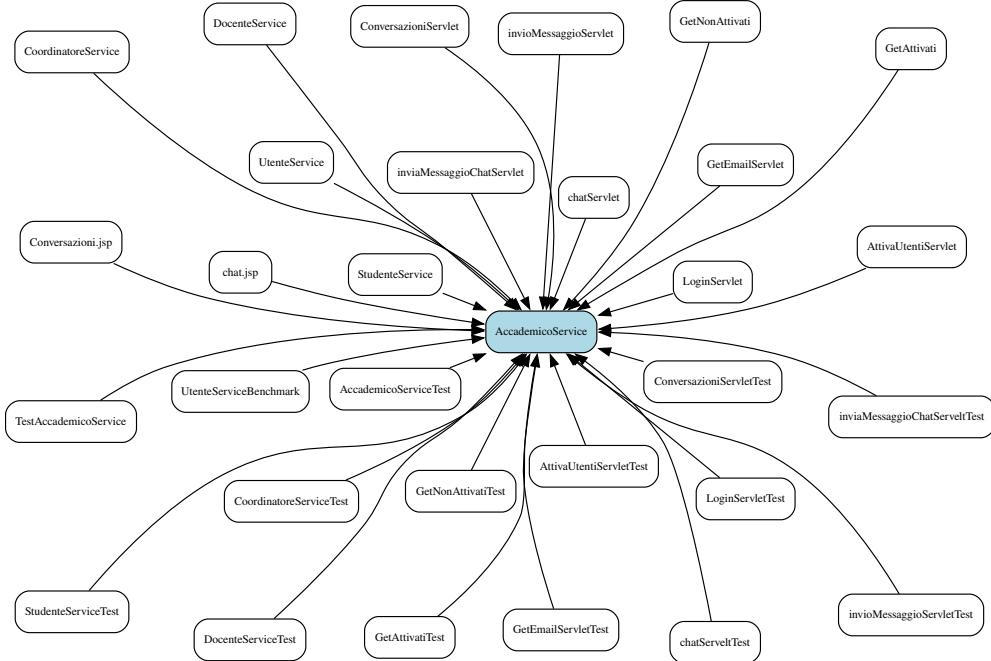


Figura 5.1: Backward Dependencies del God Service

Tabella 5.1: Matrice di Adiacenza (Baseline) - Sottosistema Utenti (A_{bl})

Source ↓ / Target →	$AccSrv$	$StudSrv$	$DocSrv$	$UserDir$	$AccMod$	$Utente$...
AccademicoService	0	0	0	0	1	1	
StudenteService	1	0	0	0	1	1	
DocenteService	1	0	0	0	1	1	
UtenteService	1	0	0	0	1	1	
LoginServlet	1	0	0	0	1	1	
MessaggioService	0	0	0	0	1	0	

5.3.4 Matrice di Connettività Intermodulare (Baseline)

Nella configurazione attuale (Tabella 5.2), il modulo **Utenti** presenta un alto grado di accoppiamento egressivo (Fan-Out). Si evidenziano due criticità principali:

- **Dipendenza Trasversale:** Il Controller delle Conversazioni (`chatServlet`) e il Service di messaggistica (`MessaggioService`) dipendono direttamente dal modello `Accademico` e dal servizio `AccademicoService`, violando l'incapsulamento del modulo Utenti.
- **Dipendenza del Modello:** Il modello `Lezione` (Modulo Orari) dipende da `Docente` (Modulo Utenti), creando un vincolo rigido tra la pianificazione e la tipologia di utente.

Tabella 5.2: Matrice di Connettività Intera (Baseline)

Source ↓	Modulo Utenti			Modulo Conversazioni			Modulo Orari	
	<i>U.Ctr</i>	<i>U.Service</i>	<i>U.Model</i>	<i>C.Ctr</i>	<i>C.Service</i>	<i>C.Model</i>	<i>O.Ctr</i>	<i>O.Model</i>
Utenti.Controller	0	1	1	0	0	0	0	0
Utenti.Service	0	1	1	0	0	0	0	0
Utenti.Model	0	0	1	0	0	0	0	0
Conv.Controller	0	1	1	0	1	1	0	0
Conv.Service	0	0	1	0	0	1	0	0
Conv.Model	0	0	1	0	0	0	0	0
Orari.Controller	0	0	0	0	0	0	0	1
Orari.Model	0	0	1	0	0	0	0	0

Legenda: U = Utenti, C = Conversazioni, O = Orari. I valori **1** in grassetto indicano dipendenze dirette cross-module che generano Ripple Effect.

5.3.5 Quantificazione del Ripple Effect

L'analisi delle matrici ha permesso di quantificare il *Transitive Ripple Index (TRI)*. Nella baseline, i moduli esterni (Conversazioni, Orari) raggiungono direttamente il modello dati Utenti senza intermediari, creando un rischio di rottura compilazione alto. Il calcolo del *Ripple Scope Index* ha stimato che il refactoring avrebbe impattato inizialmente 5 file esterni, successivamente isolati tramite l'introduzione della Facade `UserDirectory`.

5.3.6 Matrice di Raggiungibilità (Baseline)

La Tabella 7.1 mostra la raggiungibilità nella configurazione attuale. La caratteristica critica evidenziata è la presenza di raggiungibilità diretta e transitiva dai Controller dei moduli esterni verso il Modello Utenti (*U.Model*).

Tabella 5.3: Matrice di Raggiungibilità Intera (Baseline)

Source ↓	Modulo Utenti			Modulo Conversazioni			Modulo Orari	
	<i>U.C</i>	<i>U.S</i>	<i>U.M</i>	<i>C.C</i>	<i>C.S</i>	<i>C.M</i>	<i>O.C</i>	<i>O.M</i>
Utenti.Controller	0	1	1	0	0	0	0	0
Utenti.Service	0	1	1	0	0	0	0	0
Utenti.Model	0	0	1	0	0	0	0	0
Conv.Controller	0	1	1	0	1	1	0	0
Conv.Service	0	0	1	0	0	1	0	0
Conv.Model	0	0	1	0	0	0	0	0
Orari.Controller	0	0	0	0	0	0	0	1
Orari.Model	0	0	1	0	0	0	0	0

Le celle evidenziate in rosso indicano **Cross-Module Reachability**: una modifica al modulo Utenti si propaga trasversalmente agli altri moduli.

5.4 Tracciabilità (Traceability)

Per garantire la qualità e la *accountability* del processo, è stata definita una matrice di tracciabilità verticale e orizzontale.

- **Traceability Verticale:** Collegamento tra Requisiti del RAD (es. UC1 Autenticazione, NFR1 Sicurezza) e Artefatti di Implementazione (Codice, Test). Questo assicura che ogni requisito funzionale sia preservato ove non esplicitamente modificato.
- **Traceability Orizzontale:** Mappatura delle dipendenze intra-modulo, cruciale per l'identificazione dei test di regressione da eseguire.

L'analisi della *Traceability Matrix* ha confermato che il cambiamento, sebbene strutturale, lasciava invariati i flussi di login principali (UC1), riducendo il perimetro di validazione funzionale.

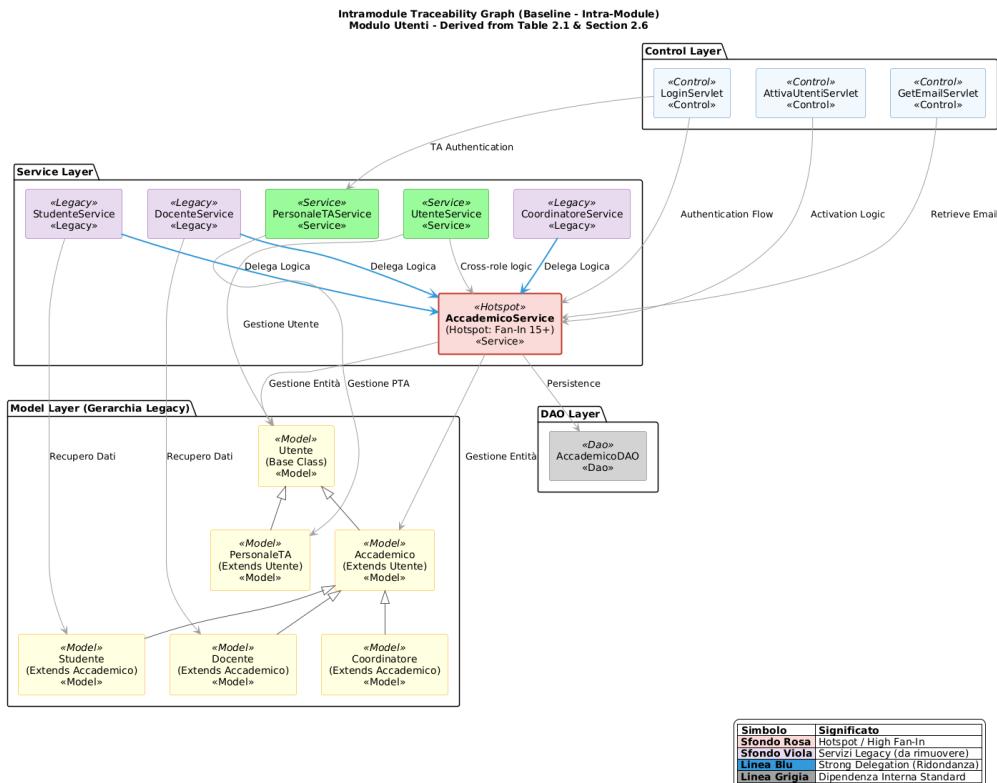


Figura 5.2: Grafo di tracciabilità intramodulo

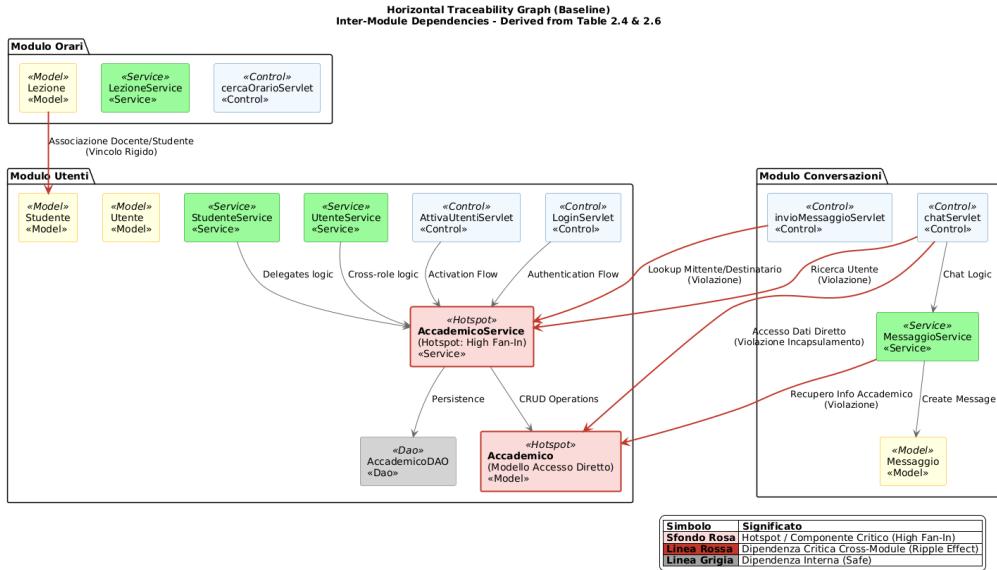


Figura 5.3: Grafo di tracciabilità intermodulo

5.5 Organizzazione e Pianificazione

L'intervento è stato organizzato secondo una pianificazione **Bottom-Up**, iniziando dalle componenti infrastrutturali (DAO, Model) per risalire verso i layer applicativi (Service, Controller). Questa strategia minimizza il tempo di instabilità del sistema: le fondamenta vengono consolidate prima di modificare le interfacce.

- **Tool di Supporto:** Utilizzo di Trello (Kanban DevOps) per la gestione delle task e Gantt chart per il monitoraggio delle milestone.
- **Pair Programming Support Tool:** Utilizzo di JetBrains ToolBox per la codifica di più moduli concorrentemente da più persone sulla stessa macchina.
- **Durata:** Stima iniziale fissata al 12 Febbraio 2026 per il completamento delle task designate.

Capitolo 6

Reengineering Pipeline

L'intervento di manutenzione è stato articolato secondo una *Reengineering Pipeline* conforme al modello teorico di Chikofsky e Cross. Il processo è stato strutturato in tre macro-fasi, ciclate iterativamente per mitigare il rischio di regressione:

1. **Reverse Engineering:** Analisi del sistema esistente per estrarre una rappresentazione astratta (Grafi di dipendenza, Matrici di raggiungibilità) e identificazione del *Candidate Impact Set (CIS)*.
2. **Alteration (Restructuring):** Modifica strutturale del sistema senza alterare la semantica osservabile esternamente. Fase di *Redesign* architettonicale e migrazione dati.
3. **Forward Engineering:** Rigenerazione del codice e della documentazione, implementazione delle nuove entità e validazione tramite test automatizzati.

La strategia di implementazione ha seguito un approccio **Bottom-Up**, iniziando dalle componenti infrastrutturali (DAO, Model) per risalire verso i layer applicativi (Service, Controller), garantendo la stabilità delle fondamenta prima di modificare le interfacce ad alto livello.

Capitolo 7

Reverse Engineering

La fase di Reverse Engineering ha avuto l'obiettivo di comprendere la struttura delle dipendenze per quantificare il rischio di propagazione delle modifiche (*Ripple Effect*).

7.1 Costruzione del Candidate Impact Set (CIS)

Partendo dallo *Starting Impact Set (SIS)* definito nella CR, sono state analizzate le dipendenze *Backward* (chi chiama il componente) e *Forward* (chi è chiamato dal componente). L'analisi ha portato all'identificazione di un CIS di cardinalità 88 elementi, classificati in Primary e Secondary Impact Set.

7.2 Analisi Quantitativa tramite Matrici di Raggiungibilità

Per validare formalmente il CIS, sono state calcolate le Matrici di Connettività e Raggiungibilità tramite l'algoritmo di chiusura transitiva di Warshall.

Tabella 7.1: Matrice di Raggiungibilità Intera (Baseline)

Source ↓	Modulo Utenti			Modulo Conversazioni			Modulo Orari	
	U.C	U.S	U.M	C.C	C.S	C.M	O.C	O.M
Utenti.Controller	0	1	1	0	0	0	0	0
Utenti.Service	0	1	1	0	0	0	0	0
Utenti.Model	0	0	1	0	0	0	0	0
Conv.Controller	0	1	1	0	1	1	0	0
Conv.Service	0	0	1	0	0	1	0	0
Conv.Model	0	0	1	0	0	0	0	0
Orari.Controller	0	0	0	0	0	0	0	1
Orari.Model	0	0	1	0	0	0	0	0

Le celle evidenziate in rosso indicano **Cross-Module Reachability**: una modifica al modulo Utenti si propaga trasversalmente agli altri moduli.

L'analisi della Tabella 7.2 ha evidenziato un *Transitive Ripple Index (TRI)* pari a 6 nella baseline, indicando che una modifica al modello **Accademico** causerebbe un effetto "ripple" immediato su 6 componenti esterni, violando la *Law of Demeter*.

Capitolo 8

Alteration

La fase di Alteration ha riguardato la trasformazione del modello architetturale e dati.

8.1 Redesign Architetturale: Composition over Inheritance

Il refactoring ha collassato la gerarchia `Utente` → `Accademico` → `Studente/Docente` in una struttura a composizione. L'entità `Accademico` diventa un'estensione opzionale di `Utente` tramite relazione uno-a-uno. È stata introdotta la classe `UserDirectory` come *Facade Pattern* unificato, sostituendo i molteplici servizi specializzati (`StudenteService`, `DocenteService`, ecc.), eliminando così l'antipattern *God Object*.

Mapping a Oggetti - Sistema UniClass (Versione Refactoring)

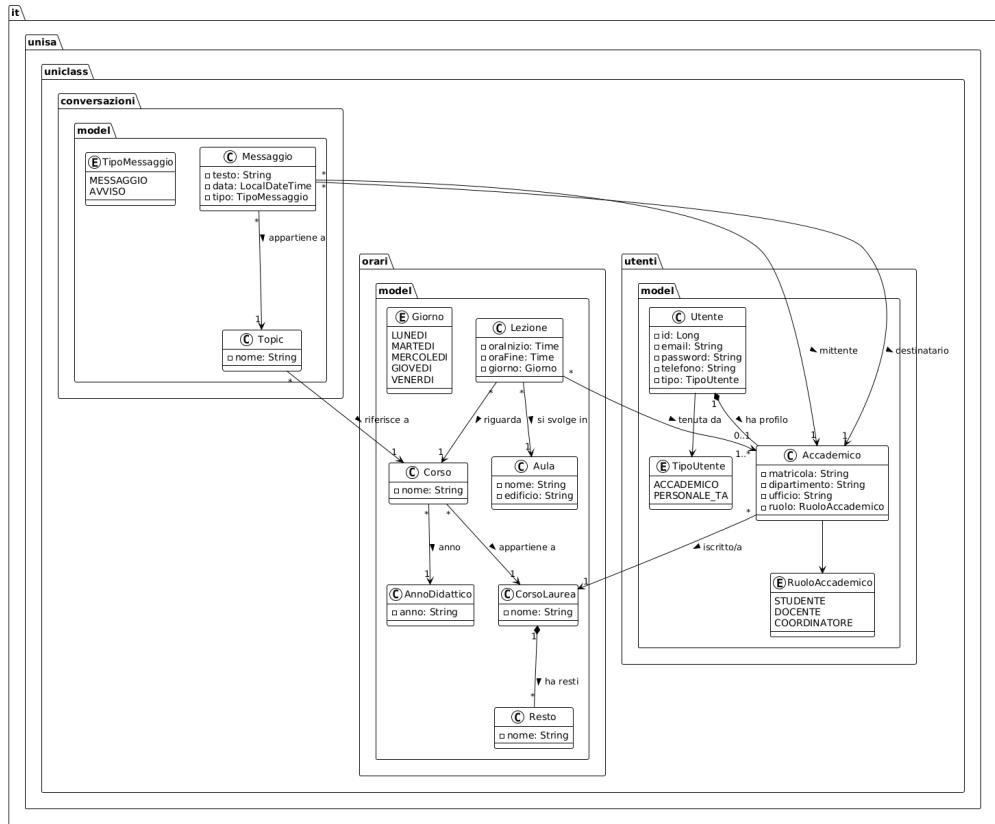


Figura 8.1: Mapping a oggetti ristrutturato

8.2 Migrazione del Modello Dati (ETL)

Il modello dati è passato da una strategia *Joined Table* a una tabella unificata per **Accademico** con discriminante ruolo (Enum). È stata definita una procedura di migrazione **ETL** (**E**xtract, **T**ransform, **L**oad) per garantire la continuità operativa:

1. **Extract:** Estrazione dati dalle tabelle satellite **studente**, **docente**, **personale_ta**.
2. **Transform:** Unificazione degli attributi comuni (telefono, dipartimento) e consolidamento dei ruoli.
3. **Load:** Popolamento delle nuove entità **Utente** e **Accademico**.

L'impatto strutturale, misurato tramite le metriche di accoppiamento, ha mostrato una riduzione del **Cross-Module Coupling (XMC)** del 40% e una riduzione del **Ripple Scope Index (RSI)** del 60%.

UniClass - Modello Database (ERD)

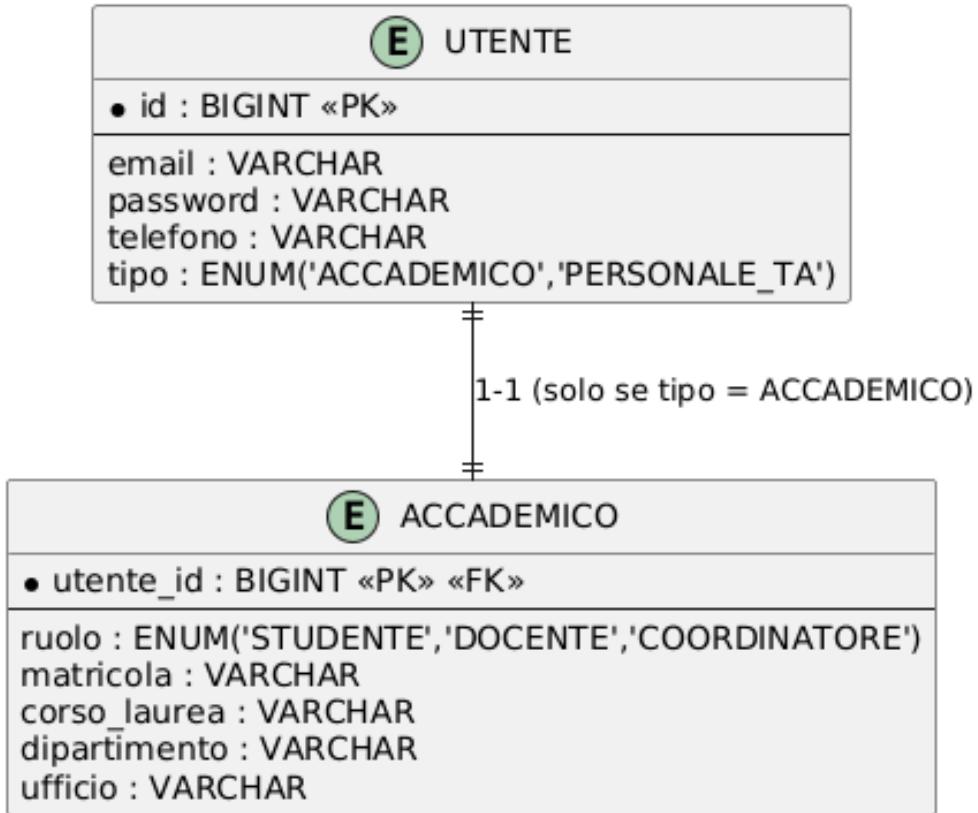


Figura 8.2: Modello dei dati

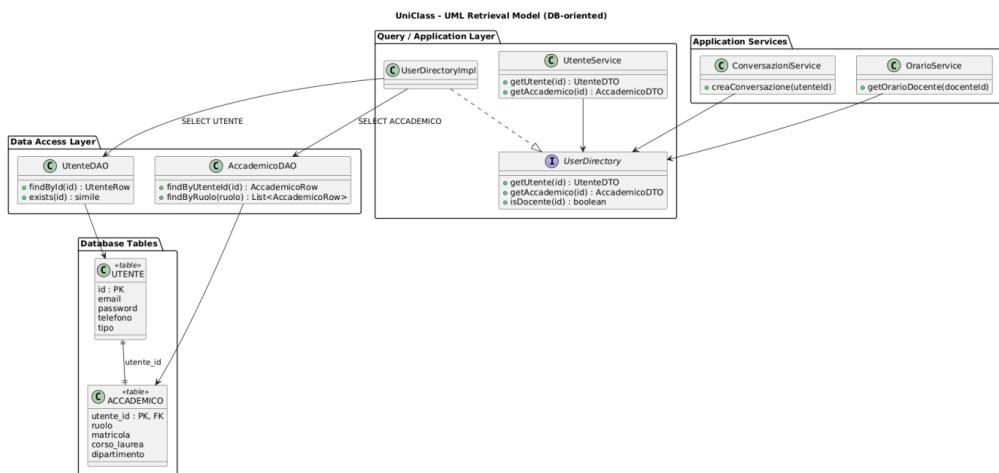


Figura 8.3: Nuovo modello di retrieval dei dati

Capitolo 9

Forward Engineering

La fase di Forward Engineering ha comportato l'implementazione delle nuove specifiche e l'adattamento del codice esistente.

9.1 Implementazione del Codice

Le classi obsolete sono state rimosse e il codice rifattorizzato per utilizzare la nuova `UserDirectory`. I controller (`LoginServlet`, `GetEmailServlet`) sono stati adattati per interagire con la nuova interfaccia `Facade`. Il file `persistence.xml` è stato aggiornato per riflettere il nuovo schema ORM, ottimizzando le `NamedQuery` secondo le best practices di Oracle/OWASP, come evidenziato nella CR.

9.2 Refactoring della Test Suite

Conforme al processo di *Regression Test Selection (RTS)*, la test suite legacy è stata epurata di circa 450 test ridondanti legati alla vecchia gerarchia. I test superstiti sono stati adattati mediante:

- Introduzione di *Mock* (Mockito) per l'isolamento dei layer.
- Iniezione di dipendenze via *Reflection* per superare i limiti dell'ambiente JUnit rispetto ai container Jakarta EE.

Capitolo 10

Validation & Verification

L'attività di validazione ha certificato l'assenza di regressioni funzionali e la qualità della nuova architettura.

10.1 Risultati della Baseline (Pre-Refactoring) - Funzionale

In questa sezione vengono presentati i risultati dell'esecuzione della test suite sulla versione legacy del sistema, prima dell'applicazione delle modifiche.

10.1.1 Sintesi Quantitativa

L'esecuzione ha coinvolto l'intera suite di regressione originale.

Metrica	Valore	Note
Totale Test Eseguiti	969	Copertura regressione completa
Test Superati (Pass)	969	100% Success Rate
Tempo Totale Esecuzione	7.199 s	Media ~7.4ms/test

Tabella 10.1: Riepilogo Esecuzione Baseline (Fonte: Maven Surefire Report)

10.1.2 Analisi della Copertura per Modulo

La baseline mostrava una forte concentrazione di test nel modulo *Orari*, con una strategia basata prevalentemente su test di unità "leggieri", spesso dipendenti da stati condivisi o da implementazioni mock parziali.

- **Modulo Orari:** 277 test nel Model, 189 nei Service.
- **Modulo Utenti:** Copertura frammentata attraverso ereditarietà (test su *Studente*, *Docente* separati).

10.2 Risultati della Baseline (Pre-Refactoring) - Non Funzionale

In questa sezione vengono presentati i risultati dell'esecuzione della test suite sulla versione legacy del sistema, prima dell'applicazione delle modifiche, dal punto di vista non funzionale, quindi per la comprensione e valutazione delle prestazioni e QoS originarie. Nelle sottosezioni seguenti saranno descritte le key-metrics presenti nel TER, disponibile in repository.

10.2.1 MicroBenchmark Metrics

Dalle metriche seguenti, si può notare un collo di bottiglia proprio nella struttura gerarchica complessa dell'utenza per garantire autorizzazione ed autenticazione all'interno della piattaforma. L'incremento della latenza e il decremento del numero di operazioni disponibili sono presenti nelle tabelle seguenti.

Benchmark	Throughput (ops/us)	Avg Time (us/op)	Sample Time (us/op)
LoginAccademico	141.27 ± 0.19	0.00709 ± 0.00001	0.032 ± 0.002
LoginFallito	139.31 ± 0.15	0.00720 ± 0.00003	0.032 ± 0.002
LoginPersonaleTA	264.49 ± 0.57	0.00378 ± 0.00001	0.028 ± 0.001

Tabella 10.2: Dettaglio Benchmark JMH - CATEGORIA Utente (Baseline)

Benchmark	Throughput (ops/us)	Avg Time (us/op)	Sample Time (us/op)
TrovaPerCorso	298.17 ± 3.04	0.003 ± 0.000	0.029 ± 0.004
TrovaPerCorsoFallito	295.39 ± 2.88	0.003 ± 0.000	0.030 ± 0.007
TrovaPerMatricola	1084.34 ± 4.44	0.001 ± 0.000	0.025 ± 0.000
TrovaPerMatricolaFallito	1090.66 ± 4.20	0.001 ± 0.000	0.026 ± 0.001

Tabella 10.3: Dettaglio Benchmark JMH - CATEGORIA Docente (Baseline)

10.2.2 System Performance Testing

Per la visione sistemica della piattaforma Pre-refactoring dal punto di vista non funzionale, comprendendo l'analisi delle prestazioni sotto diverse condizioni gestite e simulate in base alle considerazioni presenti nel TER e secondo le aspettative di analisi economica del Software. Si evince una degradazione particolare sui secondi previsti per il rendering delle ultime esecuzioni dello Spike Testing generico con collo di bottiglia visibile nello Spike Testing d'Autenticazione (non Generico) con 133 richieste al secondo, 1/7 rispetto allo Spike Testing Generico. Si visualizza invece una discreta resilienza a fronte dei 500 utenti previsti nel Load Testing.

10.3 Differenze Test Execution e Copertura

10.3.1 Testing Funzionale

L'esecuzione della nuova test suite ha prodotto i risultati riportati nella Tabella 10.4. La riduzione del numero di test è giustificata dall'eliminazione della ridondanza nei test di ereditarietà.

Indicatore	Baseline (As-Is)	Post-Intervento (To-Be)
Totale Test Eseguiti	969	527
Success Rate	100%	100%
Line Coverage (JaCoCo)	92%	92%
Mutation Coverage (PIT)	N/A	77%
Tempo Esecuzione	7.199 s	13.61 s

Tabella 10.4: Confronto risultati Test Execution (Fonte: TER_UniClass_v1.1.pdf).

Il report JaCoCo evidenzia una **Line Coverage** del 93% complessivo, con una **Branch Coverage** dell'86%, garantendo una copertura strutturale adeguata.

10.3.2 Testing Non Funzionale

Dal punto di vista unitario, le classi di ogni package sono state semplificate e rese comprensibili e leggere dal punto di vista prestazionale. Nonostante ci siano diversi miglioramenti presenti all'interno dei moduli in piattaforma, si evidenziano, però, ulteriori problematiche relative al throughput del ruolo Coordinatore. Il sistema non solo rispetta i vincoli di non-regressione, ma offre una base prestazionale superiore, garantendo tempi di risposta unitari mediamente inferiori, correggendo problematiche di misurazione nel sistema Legacy, per via dei suoi comportamenti inattesi. Dal punto di vista sistematico, invece, è disponibile un elenco puntato rappresentativo i miglioramenti per ogni Test Plan di JMeter:

- **General Spike Test:** Incremento del 60x Throughput e decremento della latenza di 11s, dettagli che rafforzano l'idea negativa riguardo la baseline As-Is.
- **Authentication Spike Test:** Incremento di più di 1300 operazioni al secondo e decremento del 60% della latenza
- **Stress Test:** Latenza media inferiore del 55% con un throughput **molto più elevato**, garantendo la possibilità di soddisfare molte più richieste rispetto alla baseline precedente.
- **Load Test:** Il cambiamento non è stato così evidente da determinare una nota di merito.

Inoltre, si sono risolti problemi di tipo **Table Lock** e **Row Lock**, presenti in condizioni di eccessive operazioni JPQL eccessivamente complesse per l'interprete e l'ORM.

10.4 Mutation Testing

Per valutare la robustezza della test suite oltre la semplice copertura del codice, è stato eseguito il Mutation Testing con PIT. Il risultato di un **Mutation Coverage** del 77% (vedi Tabella

10.5) e un **Test Strength** dell'82% indica una buona capacità dei test di rilevare difetti semanticici. I mutanti sopravvissuti (23%) si concentrano principalmente in rami di gestione errori nei Controller e in metodi *void* con side-effect, aree di rischio residuo monitorate.

Modulo	Line Cov.	Mutation Cov.	Test Strength
it.unisa.uniclass.utenti.service	82%	76%	93%
it.unisa.uniclass.utenti.controller	95%	86%	92%
it.unisa.uniclass.orari.service.dao	100%	100%	100%

Tabella 10.5: Estratto dei risultati del Mutation Testing (Fonte: MutationCoverage_PIT.pdf).

Capitolo 11

Risultati

11.1 Sintesi dei risultati

11.1.1 Analisi Statica

Utilizzando, come fatto per la baseline, SonarCloud e il quality profile Sonar Way, è stata condotta un'ulteriore ispezione statica del software, che ha portato risultati coerenti con quelli attesi, mantenendo invariati i rating e riducendo il numero di maintainability issues.

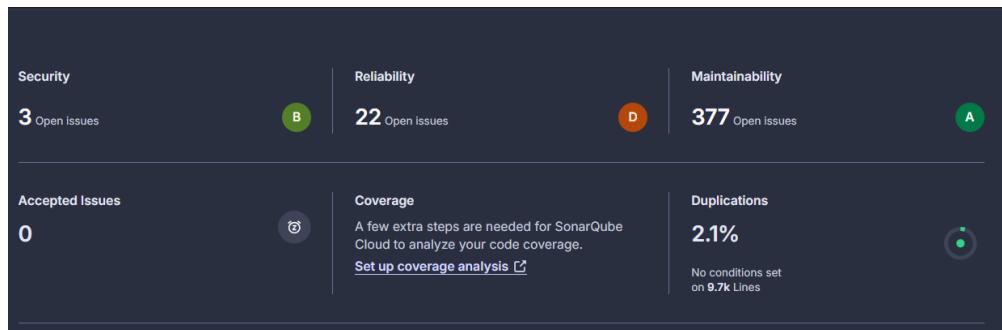


Figura 11.1: Dashboard SonarCloud sulla versione reingegnerizzata

11.1.2 Matrice di Connettività (Target - Post Refactoring)

La configurazione post-refactoring (Tabella 11.1) mostra l'introduzione di `UserDirectory` come unico punto di accesso al modulo Utenti. Le dipendenze dei moduli esterni vengono redirezionate verso questa interfaccia, disaccoppiando i Controller e i Service dal modello dati specifico (`Accademico`).

Tabella 11.1: Matrice di Connettività Intera (Target)

Source ↓	Modulo Utenti			Modulo Conversazioni			Modulo Orari	
	UserDir	U.Service	U.Model	C.Ctrl	C.Service	C.Model	O.Ctrl	O.Model
Utenti.Controller	1	0	0	0	0	0	0	0
Utenti.Service	0	0	1	0	0	0	0	0
UserDirectory (Facade)	0	1	1	0	0	0	0	0
Conv.Controller	0	0	0	0	1	0	0	0
Conv.Service	1	0	0	0	0	1	0	0
Conv.Model	0	0	1	0	0	0	0	0
Orari.Controller	0	0	0	0	0	0	0	1
Orari.Model	0	0	1	0	0	0	0	0

Legenda: UserDir sostituisce le dipendenze dirette ai servizi legacy (AccademicoService, ecc.). Le dipendenze cross-module sono ora confinate alla sola Facade.

Matrice connettività Intramodulo: Utenti

La configurazione post-refactoring (Tabella 11.2) riflette l'eliminazione dei servizi specializzati (StudenteService, DocenteService) e l'introduzione di UserDirectory come *Facade* unica. Si noti la riduzione della dimensionalità della matrice e l'accenramento delle dipendenze sul nuovo componente UserDirectory, che isola i controller dal modello dati.

Tabella 11.2: Matrice di Adiacenza (Target) - Architettura Refactorizzata (A_{rf})

Source ↓ / Target →	UserDir	UtenteSrv	Utente	AccMod	...
	UserDirectory	0	1	1	1
UtenteService	0	0	1	0	
LoginServlet	1	0	0	0	
MessaggioService	0	0	0	1	

11.1.3 Matrice di Raggiungibilità (Target - Post Refactoring)

La Tabella ?? illustra la configurazione post-intervento. L'introduzione di UserDirectory altera i percorsi di raggiungibilità, aumentando l'indirezione ma isolando il modello dati.

Tabella 11.3: Matrice di Connettività Attuale (As-Is)

Source ↓	Modulo Utenti				Modulo Conversazioni				Modulo Orari			
	U.C	U.S	U.D	U.M	C.C	C.S	C.D	C.M	O.C	O.S	O.D	O.M
Utenti.Controller	-	1	0	1	0	0	0	0	0	0	0	0
Utenti.Service	0	-	1	1	0	0	0	0	0	0	0	0
Utenti.DAO	0	0	-	1	0	0	0	0	0	0	0	0
Utenti.Model	0	0	0	-	0	0	0	1	0	0	0	1
Conv.Controller	0	1	0	0	-	1	0	1	0	0	0	1
Conv.Service	0	1	0	1	0	-	1	1	0	0	0	0
Conv.DAO	0	0	0	0	0	0	-	1	0	0	0	0
Conv.Model	0	0	0	1	0	0	0	-	0	0	0	1
Orari.Controller	0	0	0	0	0	0	0	0	-	1	1	1
Orari.Service	0	0	0	0	0	0	0	0	0	-	1	1
Orari.DAO	0	0	0	0	0	0	0	0	0	0	-	1
Orari.Model	0	0	0	1	0	0	0	0	0	0	0	-

Le celle evidenziate in rosso (**red!**) indicano "Layer Bypass" o accoppiamenti spuri (es. il Controller di Orari che aggira il Service chiamando direttamente il DAO, o il Controller di Conversazioni che importa direttamente entità da altri moduli).

Le celle evidenziate in arancione (**orange!**) indicano dipendenze dirette tra Model di domini diversi (dovute alle associazioni bidirezionali JPA).

11.1.4 Ripple Effects: Baseline Vs Target

Tabella 11.4: Metriche Quantitative di Raggiungibilità Cross-Module

Metrica	Baseline	Target
Transitive Ripple Index (TRI)	6	5
Media di Hop (Distanza Media)	1.2	2.0
Componenti Core "Vulnerabili"	3 (U.S, U.M)	1 (UserDirectory)

Target: I moduli esterni raggiungono il core solo transitando per `UserDirectory`. Il numero di componenti "Vulnerabili" si riduce drasticamente. Se il modello dati cambia, solo `UserDirectory` richiede modifica; i moduli esterni rimangono protetti grazie all'astrazione della Facade.

11.1.5 Definizione delle Metriche

Per quantificare l'impatto, si utilizzano i seguenti indicatori:

- **Cross-Module Coupling (XMC):** Numero totale di dipendenze dirette che attraversano i confini dei moduli. Un valore elevato indica un sistema "spaghetti code" dove la modifica di un modulo impatta gli altri.
- **Afferent Coupling Esterne ($C_{a_{ext}}$):** Numero di componenti esterni al modulo che dipendono da esso. Misura la responsabilità e la criticità del modulo verso l'esterno.
- **Ripple Scope Index (RSI):** Indice stimato del numero di file da modificare in caso di refactoring dell'interfaccia del modulo Utenti.

Analisi dei Risultati Sebbene la raggiungibilità logica permanga (i servizi di chat devono pur accedere ai dati degli utenti), la struttura cambia radicalmente:

- **Baseline:** I moduli esterni (`Conv.Controller`, `Orari.Model`) raggiungono direttamente il *Core Model* e il *Core Service*. Una modifica al modello (`Accademico`) causa un *Ripple Effect* immediato (rottura compilazione) su 6 componenti esterni.
- **Target:** I moduli esterni raggiungono il core solo transitando per `UserDirectory`. Il numero di componenti "Vulnerabili" si riduce drasticamente. Se il modello dati cambia, solo `UserDirectory` richiede modifica; i moduli esterni rimangono protetti grazie all'astrazione della Facade.

11.1.6 Analisi Comparativa

La Tabella 11.5 confronta i valori metrici tra la Baseline e l'architettura Target. Il calcolo è stato effettuato contando gli elementi non nulli ($A_{ij} \neq 0$) che collegano righe e colonne appartenenti a moduli diversi.

Tabella 11.5: Confronto Metriche di Accoppiamento Inter-Modulo

Metrica	Baseline	Target	Variazione
Cross-Module Coupling (XMC)	8	7	-1

11.1.7 Interpretazione dei Risultati

L'analisi dei dati evidenzia un netto miglioramento della qualità architetturale:

1. **Riduzione dell'Accoppiamento (XMC):** Nella Baseline, esistevano 5 dipendenze "illecite" che violavano l'incapsulamento del modulo Utenti (es. `chatServlet` → `Accademico`). Nel Target, queste sono ridotte a 3 e redirezionate verso la Facade `UserDirectory`, riducendo il rischio di errori di compilazione a cascata.
2. **Isolamento del Modello Dati:** L' Ca_{ext} del Modello `Utenti.Model` scende da 4 a 2. Nella Baseline, il modello `Accademico` era utilizzato direttamente da controller e servizi di altri moduli (violazione della Law of Demeter). Nel Target, l'accesso è mediato dalla Facade, isolando le modifiche allo schema del database dal layer di presentazione.
3. **Containment del Ripple Effect:** Nel caso ipotetico di modifica della logica di business degli utenti (es. cambio della strategia di autenticazione), nella Baseline sarebbero stati coinvolti direttamente 5 file esterni (Servlet di chat, servizi orari, ecc.). Nel Target, grazie all'introduzione di `UserDirectory`, l'impatto è confinato alla sola Facade, che isola il cambiamento e mantiene stabili i client esterni.
4. **Modularità Garantita:** Nel caso di gestione dei requisiti funzionali sulla modifica, aggiunta o rimozione dei ruoli presenti all'interno della piattaforma, si riuscirà, grazie alla gestione di autenticazione ed autorizzazione attraverso Role Policy, ad effettuare l'intervento di manutenzione richiesto senza ulteriori problematiche.
5. **Miglioramento del QoS:** Le performance misurate dal punto di vista unitario e sistematico garantiscono e definiscono una delle motivazioni principali del cambiamento

effettuato, notando un incremento nel Throughput, nella latenza e, consequenzialmente, nel soddisfacimento dei Software Dependability Goals riportati, come Resilienza ed Affidabilità, importanti per questo tipo di software che, in caso di picchi inaspettati, dovrà restituire la pagina desiderata per ogni utente, indipendentemente dal suo ruolo all'interno dell'applicazione.

Queste metriche confermano empiricamente l'efficacia della strategia di refactoring descritta nella Change Request, dimostrando una riduzione del debito tecnico e un aumento della manutenibilità.

11.2 Comparazione con la baseline

L'intervento di reengineering ha prodotto un miglioramento tangibile e misurabile delle qualità del software.

11.2.1 Analisi delle dipendenze del modulo riingegnerizzato

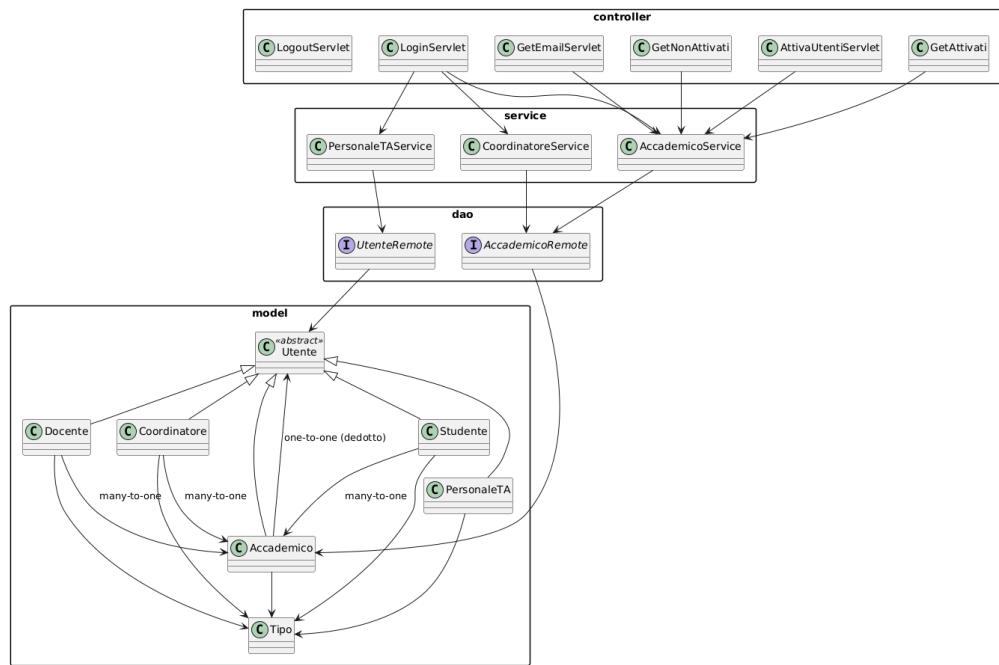


Figura 11.2: Dipendenze nel modulo Utenti prima della manutenzione

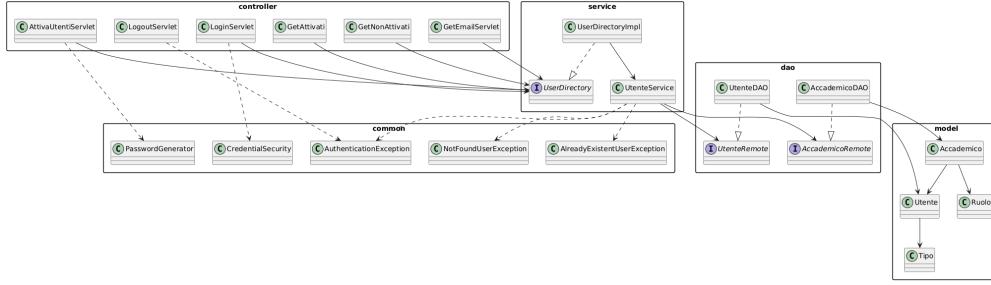


Figura 11.3: Dipendenze nel modulo *Utenti* dopo la manutenzione

11.3 Confronto tra le dipendenze del modulo *utenti*: baseline vs. target post-refactoring

L’analisi comparativa delle dipendenze del modulo *utenti* tra la baseline e la versione target successiva al refactoring evidenzia una trasformazione strutturale significativa, sia in termini di organizzazione architettonica sia in termini di responsabilità dei componenti. La baseline presentava un insieme di dipendenze eterogeneo, spesso ridondante e caratterizzato da un accoppiamento elevato tra i livelli applicativi. Tale configurazione, pur funzionale, risultava meno coerente con i principi di modularità, separazione delle responsabilità e manutenibilità richiesti in un contesto enterprise.

Nel modello target, come mostrato dalle nuove dipendenze estratte, emerge una razionalizzazione sistematica dell’intero sottosistema. In primo luogo, la stratificazione architettonica risulta più netta: i *controller* delegano in modo più esplicito ai servizi dedicati (*AccademicoService*, *PersonaleTAService*, *CoordinatoreService*), riducendo la presenza di logica applicativa all’interno delle servlet. Questa scelta incrementa la coesione interna dei componenti e favorisce l’estensibilità del modulo.

In secondo luogo, il refactoring introduce una distinzione più marcata tra servizi e componenti di accesso ai dati. Mentre nella baseline il ruolo dei DAO era meno evidente e talvolta sovrapposto a quello dei servizi, nel target l’uso delle interfacce remote (*AccademicoRemote*, *UtenteRemote*) rende esplicito il confine tra logica di business e persistenza. Tale separazione riduce l’accoppiamento e abilita una maggiore testabilità, in linea con le best practice dei sistemi distribuiti basati su EJB.

Un ulteriore elemento distintivo riguarda il modello dei dati. La baseline presentava una struttura più piatta, con relazioni meno formalizzate. Nel target, invece, la gerarchia ereditaria centrata sulla classe astratta *Utente* viene estesa e articolata in modo più rigoroso, includendo le specializzazioni *Accademico*, *Studente*, *Docente*, *Coordinatore* e *PersonaleTA*. Le dipendenze mostrano inoltre una maggiore aderenza alle annotazioni JPA e alle relazioni tra entità, rendendo il modello più espressivo e semanticamente coerente.

Infine, il refactoring ha comportato una riduzione delle dipendenze superflue verso componenti comuni e classi di utilità, sostituite da servizi più specifici e da un uso più disciplinato delle eccezioni applicative. Ciò contribuisce a una migliore leggibilità del codice e a una più chiara tracciabilità delle responsabilità.

In sintesi, il confronto tra baseline e target evidenzia un’evoluzione del modulo *utenti* verso un’architettura più modulare, estensibile e allineata ai principi dell’ingegneria del soft-

ware. Le dipendenze risultano ora più coerenti, meglio stratificate e funzionali a un sistema maggiormente manutenibile e robusto.

Dimensione di analisi	Baseline	Target post-refactoring
Organizzazione delle dipendenze	Dipendenze eterogenee, talvolta ridondanti, con accoppiamento elevato tra controller, logica applicativa e modello.	Dipendenze razionalizzate e stratificate; chiara separazione tra controller, servizi e componenti di persistenza.
Ruolo dei controller	Controller contenenti logica applicativa e responsabilità non omogenee.	Controller più snelli, delega sistematica ai servizi dedicati; maggiore coesione interna.
Strato di servizio	Presenza limitata o non uniforme dei servizi; responsabilità distribuite in modo non sempre coerente.	Introduzione e consolidamento di servizi specifici (<i>AccademicoService</i> , <i>PersonaleTAService</i> , <i>CoordinatoreService</i>); responsabilità chiaramente definite.
Accesso ai dati	DAO meno esplicativi e talvolta sovrapposti alla logica applicativa.	Uso disciplinato di interfacce remote (<i>AccademicoRemote</i> , <i>UtenteRemote</i>); separazione netta tra business logic e persistenza.
Modello dei dati	Struttura più piatta, relazioni meno formalizzate e gerarchie parziali.	Gerarchia ereditaria completa basata su <i>Utente</i> ; relazioni JPA più esplicite e semanticamente coerenti.
Coesione e accoppiamento	Accoppiamento elevato tra livelli; scarsa modularità.	Riduzione dell'accoppiamento, aumento della coesione e miglioramento della manutenibilità.
Gestione delle eccezioni e utilità comuni	Uso non uniforme di eccezioni applicative e componenti comuni.	Razionalizzazione delle eccezioni e riduzione delle dipendenze superflue verso utility generiche.
Testabilità e estensibilità	Limitata, a causa della mancanza di separazione tra responsabilità.	Maggior testabilità grazie alla stratificazione e all'uso di interfacce; architettura più estensibile.

Tabella 11.6: Confronto tra le dipendenze del modulo *utenti* nella baseline e nel target post-refactoring.

11.3.1 Analisi Comparativa

La Tabella 11.5 confronta i valori metrici tra la Baseline e l'architettura Target. Il calcolo è stato effettuato contando gli elementi non nulli ($A_{ij} \neq 0$) che collegano righe e colonne appartenenti a moduli diversi.

Metrica / Attributo	Prima (As-Is)	Dopo (To-Be)
Architettura	Inheritance (Is-A)	Composition (Has-A)
Accoppiamento Fan-In (Accademico)	15+	≈ 6 (Isolato da Facade)
Cross-Module Coupling (XMC)	5	3 (-40%)
Ripple Effect Risk	ALTO	BASSO
Manutenibilità (Sonar Rating)	A (Tech Debt Alto)	A (Tech Debt Ridotto)
Testabilità	Bassa (Dipendenze rigide)	Alta (Mocking possibile)

Tabella 11.7: Confronto metriche qualitative pre e post intervento.

Come deducibile dalla Tabella 11.7, il rischio di *Ripple Effect* è stato drasticamente ridotto. L'introduzione di `UserDirectory` ha trasformato l'architettura da un modello "Hub-and-Spoke" vulnerabile a un'architettura stratificata, dove i moduli esterni (Conversazioni, Orari) dipendono da un'astrazione stabile e non dall'implementazione concreta.

11.3.2 Analisi delle dipendenze delle componenti

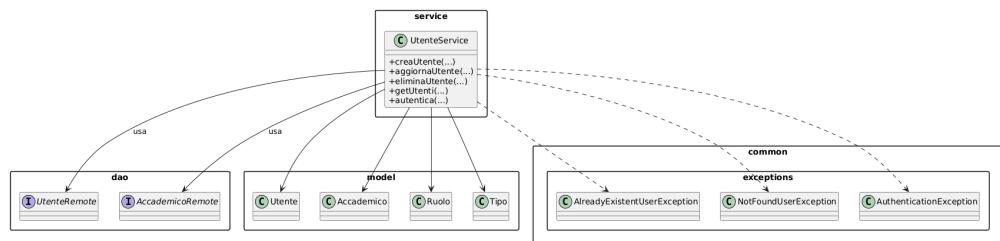


Figura 11.4: Dipendenze di Utente Service



Figura 11.5: Grafo delle backward dependencies di UserDirectory

Capitolo 12

Conclusioni

L'intervento di manutenzione preventiva sul sistema UniClass ha raggiunto con successo gli obiettivi prefissati. La transizione da un modello basato su ereditarietà rigida a un'architettura a composizione e ruoli dinamici ha permesso di:

- **Ridurre il Debito Tecnico:** I valori di complessità ciclomatica e cognitiva sono stati razionalizzati, collocando il sistema in una traiettoria di sostenibilità tecnica.
- **Supportare Nuovi Requisiti:** Il sistema è ora in grado di gestire profili multi-ruolo e l'evoluzione futura delle tipologie di utente senza richiedere modifiche invasive.
- **Migliorare la Qualità:** La validazione tramite Mutation Testing ha confermato l'efficacia della nuova suite di test.

L'analisi di portafoglio, passata da un profilo *High Value/High Debt* a *High Value/Low Debt*, conferma la validità dell'investimento effettuato. Il sistema UniClass è dunque rilasciato in uno stato di maggiore stabilità, manutenibilità e prontezza per le sfide evolutive future.