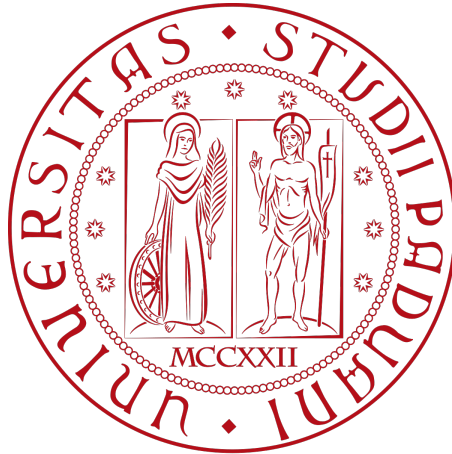


Università degli Studi di Padova

Dipartimento di Matematica «Tullio Levi-Civita»

Corso di Laurea in Informatica



**Archeologia Digitale e Rinascimento del Codice:
Modernizzazione dei Sistemi Legacy attraverso la
Migrazione Automatizzata COBOL - Java**

Tesi di laurea

Relatore

Prof. Tullio Vardanega

Laureando

Annalisa Egidi

Matricola: 1216745

Anno Accademico 2024/2025

Ringraziamenti

Sommario

L'elaborato descrive i processi, gli strumenti e le metodologie coinvolte nello sviluppo di un sistema di migrazione automatizzata per la modernizzazione di sistemi *legacy*, in particolare sulla conversione di applicazioni COBOL verso Java.

Nel dominio applicativo di interesse dell'elaborato:

- **Migrazione automatizzata:** è il processo di conversione di sistemi informatici da tecnologie obsolete a moderne architetture, preservando la logica di *business* originale;
- **Legacy Systems:** sistemi informatici datati ma ancora operativi, spesso critici per le organizzazioni, difficili da mantenere e integrare con tecnologie moderne (ingl. *legacy systems*).

Il progetto, sviluppato nel corso del tirocinio presso l'azienda Miriade Srl (d'ora in avanti **Miriade**), ha la peculiarità di aver esplorato inizialmente un approccio tradizionale basato su *parsing* deterministico per poi scegliere una soluzione innovativa basata su intelligenza artificiale generativa, dimostrando come l'AI possa cambiare drasticamente i tempi e la qualità dei risultati nel campo della modernizzazione *software*.

Struttura del testo

Il corpo principale della relazione è suddiviso in 4 capitoli:

Il **primo capitolo** descrive il contesto aziendale in cui sono state svolte le attività di tirocinio curricolare, presentando Miriade come ecosistema di innovazione tecnologica e analizzando le metodologie e tecnologie all'avanguardia adottate dall'azienda;

Il **secondo capitolo** approfondisce il progetto di migrazione COBOL - Java, delineando il contesto di attualità dei sistemi *legacy*, gli obiettivi del progetto e le sfide tecniche identificate nella modernizzazione di applicazioni COBOL verso architetture Java moderne;

Il **terzo capitolo** descrive lo sviluppo del progetto seguendo un approccio cronologico, dal *parser* tradizionale iniziale al *pivot* verso l'intelligenza artificiale, documentando le metodologie di lavoro, i risultati raggiunti e l'impatto trasformativo dell'AI sui tempi di sviluppo;

Il **quarto capitolo** sviluppa una retrospettiva sul progetto, analizzando le *lessons learned*, il valore dell'AI come *game changer* nella modernizzazione *software*, la crescita professionale acquisita e le prospettive future di evoluzione della soluzione sviluppata.

Le **appendici** completano l'elaborato con:

- **Acronimi:** elenco alfabetico degli acronimi utilizzati nel testo con le relative espansioni;
- **Glossario:** definizioni dei termini tecnici e specialistici impiegati nella trattazione;
- **Sitografia:** fonti consultate e riferimenti sitografici utilizzati per la redazione dell'elaborato.

Indice

Miriade: un ecosistema di innovazione tecnologica	1
L'azienda nel panorama informatico e sociale	1
Metodologie e tecnologie all'avanguardia	1
Architettura organizzativa	3
Investimento nel capitale umano e nella ricerca	5
Il progetto di migrazione COBOL-Java	8
Contesto di attualità	8
Obiettivi dello stage	10
Obiettivi principali	10
Obiettivi operativi	10
Metriche di successo	11
Vincoli	12
Pianificazione concordata	12
Valore strategico per l'azienda	13
Aspettative personali	14
Sviluppo del progetto: dal <i>parser</i> tradizionale all'AI (AI)	16
<i>Setup</i> iniziale e metodologia di lavoro	16
Implementazione della metodologia <i>Agile</i>	16
Strumenti di sviluppo e ambiente tecnologico	16
Gestione del progetto attraverso <i>Jira</i> e <i>Confluence</i>	17
Primo periodo: immersione nel mondo <i>COBOL</i>	18
Studio del linguaggio e creazione progetti <i>test</i>	18
Mappatura dei <i>pattern</i> e analisi di traducibilità	18
Valutazione delle soluzioni esistenti	19
Secondo periodo: sviluppo del <i>parser</i> tradizionale	19
Implementazione del <i>parser</i> Java	19
Analisi critica e limiti dell'approccio	20
Terzo periodo: <i>pivot</i> verso l'intelligenza artificiale	20
Valutazione delle <i>API</i> di AI generativa	20
<i>Design</i> del sistema AI-powered	21
Quarto periodo: implementazione della soluzione AI-driven	21
Sviluppo del <i>prompt engineering</i>	21

Implementazione del <i>translator</i> completo	21
Generazione automatica di progetti <i>Maven</i>	21
Risultati raggiunti	22
Impatto dell'AI sui tempi di sviluppo	22
Analisi qualitativa dei risultati	22
Risultati quantitativi	22
Conclusioni del capitolo	23
Valutazioni retrospettive e prospettive future	24
Analisi retrospettiva del percorso	24
L'AI come <i>game changer</i> nella modernizzazione <i>software</i>	24
Crescita professionale e competenze acquisite	24
Valore della formazione universitaria nell'era dell'AI	24
<i>Roadmap</i> evolutiva e opportunità di sviluppo	24
Lista degli acronimi	25
Glossario	26
Sitografia	28

Elenco delle figure

Figura 1	Ecosistema Atlassian - dashboard Jira	2
Figura 2	Ecosistema Atlassian - dashboard Confluence	3
Figura 3	Struttura organizzativa delle divisioni Miriade	4
Figura 4	Funzioni nella sezione Analytics	5
Figura 5	Impegni etici e morali aziendali di Miriade	6
Figura 6	Interfaccia utente e codice COBOL tipici dei sistemi legacy	8
Figura 7	Confronto tra architettura monolitica dei mainframe e architettura moderna a microservizi	9
Figura 8	Diagramma di Gantt della pianificazione del progetto	13
Figura 9	Rappresentazione della metodologia Agile applicata al progetto	15
Figura 10	Utilizzo di BitBucket per il versionamento del codice	17
Figura 11	Architettura del sistema di traduzione AI-powered	21

Miriade: un ecosistema di innovazione tecnologica

Miriade, come realtà nel panorama Information Technology (IT) italiano, si distingue per il suo approccio innovativo rispetto all'ecosistema completo del dato e alle soluzioni informatiche correlate. L'azienda, che ho avuto l'opportunità di conoscere durante il mio percorso di *stage*, si caratterizza per una filosofia aziendale orientata all'innovazione continua e all'investimento nel capitale umano, elementi che la rendono un ambiente particolarmente stimolante per la crescita professionale di figure *junior*.

L'azienda nel panorama informatico e sociale

Miriade si posiziona strategicamente nel settore dell'analisi dati e delle soluzioni informatiche, operando con quattro aree funzionali principali: *Analytics*, *Data*, *System Application* e *Operation*. L'azienda ha costruito nel tempo una solida reputazione nel mercato attraverso la capacità di fornire soluzioni innovative che rispondono non solo alle esigenze tecniche dei clienti, ma che prestano particolare attenzione alle relazioni umane e alle realtà del territorio.

Ciò che distingue *Miriade* nel contesto competitivo è la sua *vision* aziendale, che integra le competenze tecnologiche con una forte responsabilità sociale. L'azienda implementa attivamente azioni a supporto di società e cooperative del territorio, dimostrando come l'innovazione tecnologica possa essere un veicolo di sviluppo sociale ed economico locale. Questa attenzione alla dimensione sociale si riflette anche nell'approccio alle risorse umane, con una particolare propensione a individuare e coltivare giovani energie fin dalle scuole e università attraverso tirocini curriculari che permettono una crescita personale durante il percorso di studi.

La clientela di Miriade spazia tra i medi e grandi clienti, includendo sia realtà del settore privato che pubblico. Questa diversificazione del portfolio clienti permette all'azienda di confrontarsi con problematiche tecnologiche variegata, mantenendo una costante spinta all'innovazione e all'adattamento delle soluzioni proposte.

Metodologie e tecnologie all'avanguardia

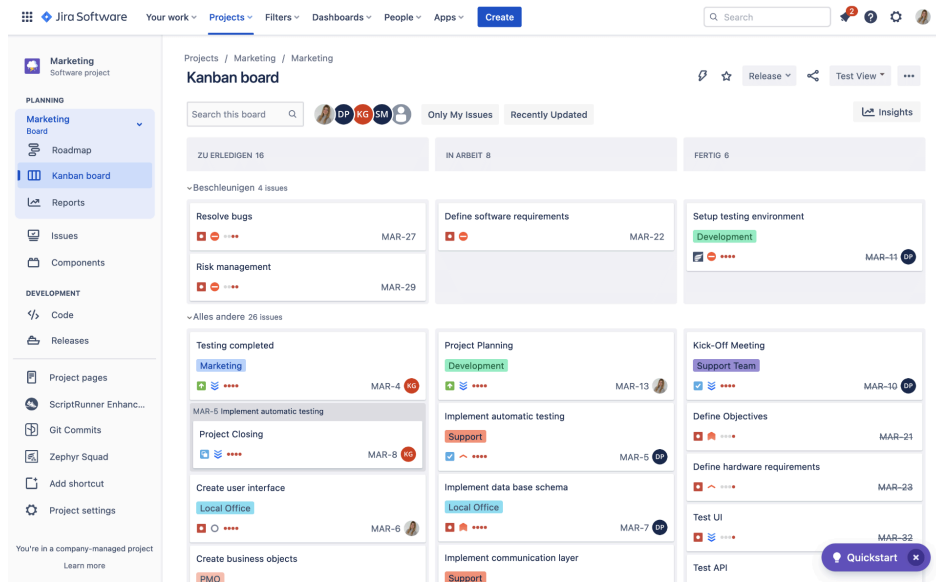
L'approccio metodologico di Miriade si fonda sull'adozione dell'*Agile* come filosofia operativa pervasiva, che permea tutti i processi aziendali e guida l'organizzazione del lavoro quotidiano. Durante il mio *stage*, ho potuto osservare direttamente come questa metodologia venga implementata attraverso *stand-up* giornalieri e *sprint* settimanali, creando un ambiente di lavoro dinamico e orientato agli obiettivi.

L'azienda utilizza sia *Kanban* che *Scrum*, adattando la metodologia alle specifiche esigenze progettuali e alle preferenze del cliente. Questa flessibilità metodologica dimostra la maturità organizzativa di Miriade e la sua capacità di adattare i processi alle diverse situazioni operative. Ho potuto constatare personalmente come gli *stand-up* mattutini fossero momenti fondamentali per l'allineamento del *team*, permettendo una comunicazione trasparente sullo stato di avanzamento delle attività e una rapida identificazione di eventuali impedimenti.

Lo *stack tecnologico* adottato riflette l'attenzione dell'azienda per gli strumenti di collaborazione e versionamento. L'*Atlassian Suite* costituisce la spina dorsale dell'infrastruttura collaborativa aziendale, utilizzata in modo strutturato e pervasivo per diverse finalità:

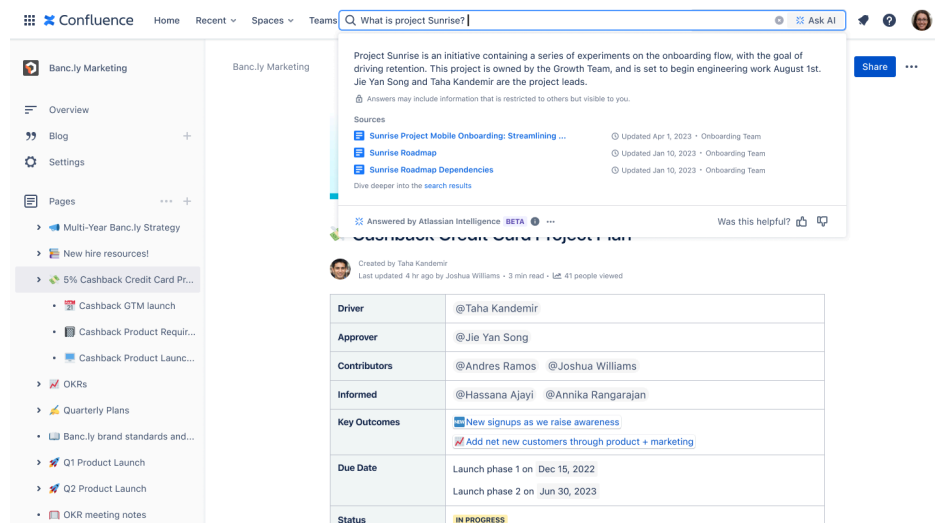
- **Confluence** per la gestione della *knowledge base* aziendale e la documentazione tecnica
- **Jira** per il *tracking* delle attività e la gestione dei progetti
- **Bitbucket** per il versionamento del codice e la collaborazione nello sviluppo

Durante il mio percorso, ho potuto apprezzare l'importanza che l'azienda attribuisce alla cultura del versionamento e della documentazione. Le Figura 1) e Figura 2 mostrano parte dell'ecosistema Atlassian integrato utilizzato quotidianamente in azienda, che ha rappresentato per me un elemento fondamentale nell'apprendimento delle pratiche professionali di sviluppo software.



Fonte: <https://www.peakforce.dev>

Figura 1: Ecosistema Atlassian - dashboard Jira



Fonte: <https://support.atlassian.com>

Figura 2: Ecosistema Atlassian - dashboard Confluence

La formazione continua sulle tecnologie emergenti è parte integrante della cultura aziendale. L'area *Analytics*, in particolare, mantiene un *focus* costante sull'esplorazione e implementazione di soluzioni basate su AI e Large Language Models (LLM), che rappresentano il naturale proseguimento di quello che precedentemente veniva incasellato come «*big data*» ed è parte integrante della strategia aziendale.

Architettura organizzativa

L'architettura organizzativa di Miriade si distingue per la sua struttura «piatta». L'azienda ha adottato un modello organizzativo che prevede solo due livelli gerarchici: l'amministratore delegato e i responsabili di area. Questa scelta strutturale facilita la comunicazione diretta e riduce le barriere comunicative, creando un ambiente di lavoro agile e responsabilizzante.

Le quattro aree funzionali principali - *Analytics*, *Data*, *System Application* e *Operation* - operano con un alto grado di autonomia, pur mantenendo una forte interconnessione attraverso aree trasversali. Queste aree trasversali, composte da persone provenienti dalle diverse divisioni, si occupano di attività di innovazione a vari livelli, come *DevOps*, *Account Management* e *Research & Development*. Questa struttura matriciale permette una *cross-fertilizzazione* delle competenze e favorisce l'innovazione continua. La rappresentazione visuale in Figura 3 illustra chiaramente questa struttura organizzativa interconnessa.



Figura 3: Struttura organizzativa delle divisioni Miriade

La divisione *Analytics*, nella quale ho avuto il piacere di lavorare, guidata da Arianna Bellino, conta attualmente 17 persone ed è in veloce crescita. Rappresenta il motore di innovazione dell'azienda, specializzandosi nella gestione del dato, dal dato grezzo all'analisi avanzata, tramite approcci e tecnologie AI e LLM *based*, con *focus* sull'automazione dei processi e alla riduzione delle attività routinarie. I membri del *team* non hanno ruoli rigidamente definiti, ma piuttosto funzioni che possono evolversi in base alle esigenze progettuali e alle competenze individuali. Ho osservato dipendenti che svolgevano funzioni diverse quali:

- Pianificazione e gestione progetti
- Attività di prevendita e consulenza
- Ricerca e sviluppo di nuove soluzioni
- Sviluppo *software* e *data analysis*

Questa fluidità organizzativa crea un ambiente stimolante dove ogni persona può contribuire in modi diversi, favorendo la crescita professionale multidisciplinare. Durante lo stage, ho potuto interagire con colleghi che ricoprivano diverse funzioni, beneficiando della loro esperienza e prospettive diverse. La varietà di funzioni all'interno della divisione *Analytics* è rappresentata in Figura 4, che evidenzia la natura dinamica e multifunzionale del team.



Figura 4: Funzioni nella sezione Analytics

Il ruolo dello stagista in questo ecosistema aziendale è particolarmente valorizzato. Non viene visto come una risorsa marginale, ma come parte integrante del team, con la possibilità di contribuire attivamente ai progetti e di proporre soluzioni innovative. Il sistema di tutoraggio è strutturato con l'assegnazione di un tutor dell'area specifica e di un mentor che può provenire anche da altre aree. Il tutor segue il percorso tecnico dello stagista, mentre il mentor fornisce supporto a livello emotivo e di inserimento aziendale.

Particolarmente apprezzabili sono gli incontri settimanali chiamati «tiramisù», dedicati ai nuovi entrati in azienda. Durante questi momenti, vengono analizzate le possibili difficoltà relazionali o comunicative riscontrate durante la settimana, con il supporto di una figura dedicata. Questo approccio dimostra l'attenzione dell'azienda non solo alla crescita tecnica, ma anche al benessere e all'integrazione dei propri collaboratori.

Investimento nel capitale umano e nella ricerca

L'investimento nel capitale umano rappresenta uno dei pilastri fondamentali della strategia aziendale di Miriade. Durante il mio stage, ho potuto constatare come l'azienda non si limiti a dichiarare l'importanza delle risorse umane, ma implementi concretamente politiche e programmi volti alla valorizzazione e crescita delle persone, come ad esempio incontri, riflessioni e azioni sulla Parità di Genere, sulla quale sono certificati come azienda.



Fonte: <https://www.miriade.it>

Figura 5: Impegni etici e morali aziendali di Miriade

Come si può osservare in Figura 5, l'azienda si esprime esplicitamente riguardo i propri valori.

Il processo di selezione riflette questa filosofia: l'azienda ricerca persone sensibili, elastiche, proattive e autonome, ponendo l'enfasi sulle caratteristiche personali piuttosto che esclusivamente sulle competenze tecniche pregresse, un approccio che permette di costruire *team* coesi e motivati, capaci di affrontare sfide tecnologiche in continua evoluzione.

I programmi di formazione continua sono strutturati e costanti. L'azienda investe significativamente nella crescita professionale dei propri dipendenti attraverso:

- Corsi di formazione tecnica su nuove tecnologie
- Certificazioni professionali
- Partecipazione a conferenze e *workshop*
- Sessioni di *knowledge sharing* interno
- Progetti di ricerca e sviluppo che permettono sperimentazione

Il rapporto consolidato con le università rappresenta un altro aspetto distintivo dell'approccio di Miriade al capitale umano. Gli *stage* non sono visti come semplici adempimenti formativi, ma come veri e propri laboratori di sperimentazione tecnologica. Nel mio caso specifico, il progetto di migrazione COBOL-Java è stato scelto appositamente per valutare le capacità di *problem solving* e apprendimento, con maggiore attenzione al processo seguito piuttosto che al solo risultato finale.

L'equilibrio tra formazione e produttività negli *stage* è gestito con attenzione. Inizialmente, lo *stage* è orientato totalmente sulla formazione, per poi evolvere gradualmente verso un bilanciamento equilibrato tra formazione e contributo produttivo quando lo stagista diventa sufficientemente autonomo. Nel mio caso però, trattandosi di *stage* curricolare per tesi,

l'intero percorso è stato focalizzato sulla formazione, permettendomi di esplorare in profondità tecnologie e metodologie senza la pressione di *deadline* produttive immediate.

L'investimento in risorse *junior* è visivamente significativo, questo approccio permette all'azienda di formare professionisti allineati con la propria cultura e metodologie.

In conclusione, Miriade si presenta come un ecosistema aziendale dove l'innovazione tecnologica e la valorizzazione del capitale umano si integrano sinergicamente. L'esperienza di stage in questo contesto ha rappresentato un'opportunità unica di crescita professionale, permettendomi di osservare e partecipare a dinamiche aziendali mature e orientate al futuro. La combinazione di una struttura organizzativa agile, metodologie all'avanguardia, forte investimento nelle persone e attenzione alla responsabilità sociale crea un ambiente ideale per affrontare le sfide tecnologiche contemporanee.

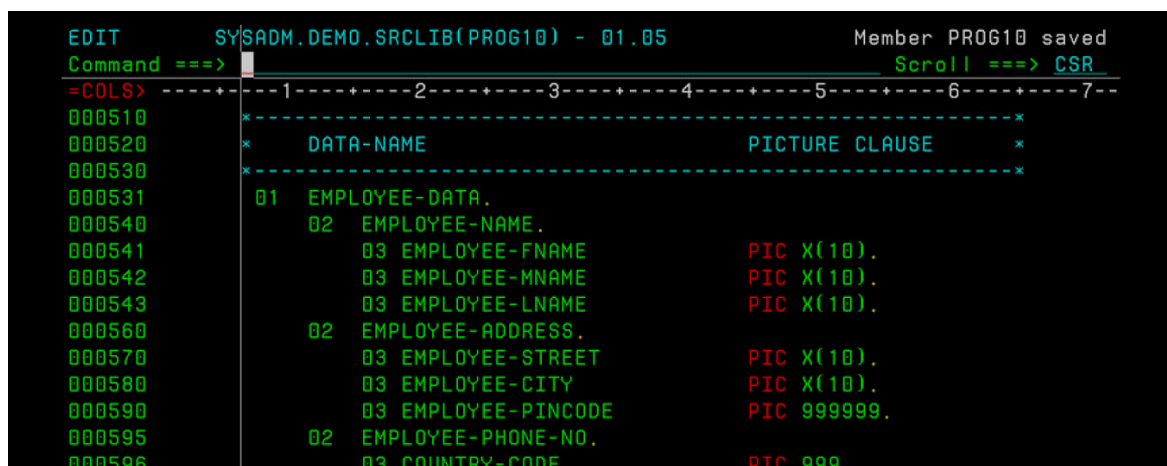
Il progetto di migrazione COBOL-Java

Il progetto di *stage* proposto da Miriade si inserisce in un contesto tecnologico di particolare rilevanza per il settore IT contemporaneo: la modernizzazione dei sistemi *legacy*. Durante il mio percorso, ho avuto l'opportunità di confrontarmi con una problematica comune a molte organizzazioni, in particolare nel settore bancario e assicurativo, dove i sistemi COBOL continuano a costituire l'impalcatura portante di infrastrutture critiche per il *business*.

Contesto di attualità

I sistemi legacy basati su Common Business-Oriented Language (COBOL) rappresentano ancora oggi una parte significativa dell'infrastruttura informatica di molte organizzazioni, specialmente nel settore bancario, finanziario e assicurativo. Nonostante COBOL sia stato sviluppato negli anni '60, ha una presenza significativa nelle moderne architetture.

Figura 6 mostra un esempio tipico di interfaccia utente e codice COBOL, che evidenzia il contrasto netto con le moderne interfacce grafiche e paradigmi di programmazione attuali. Questa differenza visuale è solo la punta dell'iceberg delle sfide che comporta il mantenimento di questi sistemi in un ecosistema tecnologico in rapida evoluzione.



```
EDIT      SYSADM.DEMO.SRCLIB(PROG10) - 01.05      Member PROG10 saved
Command ==>                                     Scroll ==> CSR
=COLS>  ---+---1---+---2---+---3---+---4---+---5---+---6---+---7---
000510      *-----*
000520      *   DATA-NAME                               PICTURE CLAUSE   *
000530      *-----*
000531      01  EMPLOYEE-DATA.
000540          02  EMPLOYEE-NAME.
000541              03  EMPLOYEE-FNAME                      PIC X(10).
000542              03  EMPLOYEE-MNAME                      PIC X(10).
000543              03  EMPLOYEE-LNAME                      PIC X(10).
000560          02  EMPLOYEE-ADDRESS.
000570              03  EMPLOYEE-STREET                    PIC X(10).
000580              03  EMPLOYEE-CITY                        PIC X(10).
000590              03  EMPLOYEE-PINCODE                    PIC 999999.
000595          02  EMPLOYEE-PHONE-NO.
000596              03  COUNTRY-CODE                      PIC 999.
```

Fonte: <https://overcast.blog>

Figura 6: Interfaccia utente e codice COBOL tipici dei sistemi legacy

La problematica della *legacy modernization* va ben oltre la semplice obsolescenza tecnologica. Durante il mio *stage*, attraverso l'analisi della letteratura e il confronto con i professionisti del settore, in particolare ho avuto modo di confrontarmi con la sig.ra Luisa Biagi, analista COBOL, ho potuto identificare come i costi nascosti del mantenimento di questi sistemi includano:

- La crescente difficoltà nel reperire sviluppatori COBOL qualificati [1]
- L'integrazione sempre più complessa con tecnologie moderne [2]
- I rischi operativi derivanti dall'utilizzo di piattaforme *hardware* e *software* che i *vendor* non supportano più attivamente [3]

Questi fattori si traducono in costi di manutenzione esponenzialmente crescenti e in una ridotta agilità nel rispondere alle esigenze di *business* in continua evoluzione.

I rischi associati al mantenimento di sistemi COBOL *legacy* nelle infrastrutture IT moderne sono molteplici e interconnessi:

- **Carenza di competenze:** La carenza di competenze specializzate crea una forte dipendenza da un *pool* sempre più ristretto di esperti, spesso prossimi al pensionamento [1].
- **Documentazione inadeguata:** La documentazione inadeguata o assente di molti di questi sistemi, sviluppati decenni fa, rende ogni intervento di manutenzione un'operazione ad alto rischio [4].
- **Incompatibilità tecnologica:** L'incompatibilità con le moderne pratiche di sviluppo come *DevOps*, *continuous integration* e *microservizi* limita in modo significativo la capacità delle organizzazioni di innovare e competere efficacemente nel mercato digitale [5].

Come illustrato in Figura 7, il contrasto tra l'architettura monolitica tipica dei sistemi *mainframe* e l'architettura moderna a microservizi evidenzia le sfide architetturali della migrazione. Questa differenza strutturale comporta non solo una riprogettazione tecnica, ma anche un ripensamento completo dei processi operativi e delle modalità di sviluppo.



Fonte: <https://www.atlassian.com>

Figura 7: Confronto tra architettura monolitica dei mainframe e architettura moderna a microservizi

La migrazione di questi sistemi verso tecnologie più moderne come Java rappresenta una sfida tecnica e una necessità strategica per garantire la continuità operativa e la competitività delle organizzazioni. Java, con il suo ecosistema maturo, la vasta *community* di sviluppatori e il supporto per paradigmi di programmazione moderni, si presenta come una delle destinazioni privilegiate per questi progetti di modernizzazione [6].

Obiettivi dello stage

Il macro-obiettivo era sviluppare un sistema prototipale di migrazione automatica da COBOL a Java che potesse dimostrare la fattibilità di automatizzare il processo di conversione, preservando la *business logic* originale e producendo codice Java idiomatrico e manutenibile.

Obiettivi principali

- **Esplorazione tecnologica:** Investigare e valutare diverse strategie di migrazione, dalla conversione sintattica diretta basata su regole deterministiche fino all'utilizzo di tecnologie di intelligenza artificiale generativa, identificando vantaggi e limitazioni di ciascun approccio.
- **Automazione del processo:** Sviluppare strumenti e metodologie che potessero automatizzare il più possibile il processo di conversione, riducendo l'intervento manuale e i conseguenti rischi di errore umano nella traduzione.
- **Qualità del risultato:** Garantire che il codice Java prodotto rispettasse standard di qualità professionale, con particolare attenzione alla leggibilità, manutenibilità e conformità alle convenzioni Java moderne.
- **Accessibilità della soluzione:** Fornire un'interfaccia utente (grafica o da linea di comando) che rendesse il sistema utilizzabile anche da personale non specializzato nella migrazione di codice.

Obiettivi operativi

Per rendere concreti e misurabili gli obiettivi principali, sono stati definiti obiettivi operativi specifici, classificati secondo tre livelli di priorità:

Obbligatori

- **OO01:** Sviluppare competenza nel linguaggio COBOL attraverso la produzione di almeno un progetto completo che includesse le quattro divisioni fondamentali (*Identification, Environment, Data e Procedure*)
- **OO02:** Esplorazione approfondita di diverse strategie di migrazione, dalla conversione sintattica diretta all'utilizzo di tecnologie di intelligenza artificiale generativa

- **OO03:** Implementare un sistema di conversione automatica che raggiungesse almeno il 75% di copertura delle divisioni
- **OO04:** Esplorare e documentare approcci distinti alla migrazione
- **OO05:** Completare la migrazione funzionante di almeno uno dei progetti COBOL sviluppati, validando l'equivalenza funzionale tra codice sorgente e risultato
- **OO06:** Produrre codice Java che rispettasse le convenzioni del linguaggio, includendo struttura dei *package*, nomenclatura standard e documentazione *JavaDoc*
- **OO07:** Fornire un'interfaccia utilizzabile (grafica o Command Line Interface (CLI)) per l'esecuzione del sistema di conversione
- **OO08:** Creare documentazione utente completa, includendo un *README* dettagliato con istruzioni di installazione, configurazione e utilizzo

Desiderabili

- **OD01:** Raggiungimento di una copertura del 100% nella conversione automatica del codice prodotto autonomamente
- **OD02:** Gestione efficace di costrutti COBOL complessi o non direttamente traducibili
- **OD03:** Implementazione di meccanismi di ottimizzazione del codice Java generato

Facoltativi

- **OF01:** Integrazione con sistemi di analisi statica per la verifica della qualità del codice generato
- **OF02:** Implementare un sistema di *reporting* dettagliato che producesse metriche sulla conversione, incluse statistiche di copertura, costrutti non convertiti e interventi manuali necessari
- **OF03:** Implementazione di funzionalità avanzate di *refactoring* del codice Java prodotto

Metriche di successo

Per valutare oggettivamente il raggiungimento degli obiettivi, erano state definite le seguenti metriche:

- **Copertura di conversione:** Percentuale di linee di codice COBOL convertite automaticamente senza intervento manuale
- **Equivalenza funzionale:** Corrispondenza interfaccia utente COBOL originale e Java convertito
- **Qualità del codice:** Conformità agli standard Java verificata tramite strumenti di analisi statica
- **Tempo di conversione:** Riduzione del tempo necessario per la migrazione rispetto a un approccio completamente manuale

- **Usabilità:** Capacità di utilizzo del sistema da parte di utenti con conoscenze base di programmazione

Vincoli

Il progetto si focalizzava sullo sviluppo di un sistema di migrazione automatica e questo aspetto caratterizzava le condizioni imposte per lo svolgimento del lavoro.

Vincoli temporali

- Durata complessiva dello *stage*: 320 ore
- Periodo: dal 05 maggio al 27 giugno 2025
- Modalità di lavoro ibrida: 2 giorni a settimana in sede, 3 giorni in modalità telematica
- Orario lavorativo: 9:00 - 18:00

Vincoli tecnologici

- Il sistema doveva essere sviluppato utilizzando tecnologie moderne e supportate
- Necessità di preservare integralmente la *business logic* contenuta nei programmi COBOL originali
- La soluzione doveva essere scalabile, capace di gestire progetti di diverse dimensioni
- Utilizzo strumenti di versionamento (*Git*) e di documentazione continua.

Vincoli metodologici

- Adozione dei principi *Agile* con *sprint* settimanali e *stand-up* giornalieri per allineamento costante
- Revisioni settimanali degli obiettivi con adattamento del piano di lavoro

Pianificazione concordata

La pianificazione del progetto seguiva un approccio flessibile, con revisioni settimanali che permettevano di adattare il percorso in base ai progressi ottenuti. La distribuzione delle attività era inizialmente stata organizzata come segue:

Prima fase - analisi e apprendimento COBOL (2 settimane - 80 ore)

- Studio approfondito del linguaggio COBOL e delle sue peculiarità
- Analisi di sistemi COBOL
- Creazione di programmi COBOL di test con complessità crescente
- Implementazione dell'interfacciamento con *database* relazionali

Seconda fase - sviluppo del sistema di migrazione (4 settimane - 160 ore)

- Analisi dei *pattern* di traduzione COBOL-Java del codice prodotto in fase precedente
- Sviluppo di uno *script* o utilizzo di *tool* esistenti per automatizzare la traduzione del codice COBOL in Java equivalente

- Gestione della traduzione dei costrutti sintattici, logica di controllo e interazioni con il *database*
- Definizione della percentuale di automazione raggiungibile e la gestione di costrutti COBOL complessi o non direttamente traducibili

Terza fase - *testing* e validazione (1 settimana - 40 ore)

- *Test* funzionali sul codice Java generato
- Confronto comportamentale con le applicazioni COBOL originali

Quarta fase - documentazione e consegna (1 settimana - 40 ore)

- Documentazione completa del sistema sviluppato
- Preparazione del materiale di consegna
- Presentazione finale dei risultati

La rappresentazione temporale dettagliata della pianificazione è visualizzata in Figura 8, che mostra la distribuzione delle attività lungo l'arco temporale dello stage.



Figura 8: Diagramma di Gantt della pianificazione del progetto

Valore strategico per l'azienda

In base a quanto ho potuto osservare e comprendere durante il periodo di *stage*, la strategia di gestione del progetto di migrazione COBOL-Java dell'azienda ospitante persegue i seguenti obiettivi:

- **Innovazione tecnologica:** l'interesse dell'azienda non era limitato allo sviluppo di una soluzione tecnica specifica, ma si estendeva all'osservazione dell'approccio metodologico e del metodo di studio che una risorsa *junior* con formazione universitaria avrebbe applicato a un problema complesso di modernizzazione IT.

- **Creazione di competenze interne:** Il progetto permetteva di sviluppare *know-how* interno su una problematica di crescente rilevanza, preparando l'azienda a potenziali progetti futuri.
- **Esplorazione di tecnologie emergenti:** Il progetto era stato concepito per esplorare la possibile applicazione dell'intelligenza artificiale generativa a problemi di modernizzazione del *software*. Questo ambito, all'intersezione tra AI e *software engineering*, può rappresentare una frontiera tecnologica di forte attualità e di interesse per un'azienda che opera già attivamente nel campo dell'AI e dei *Large Language Models*.
- **Sviluppo di *asset* riutilizzabili:** Sebbene il progetto fosse autoconclusivo, permetteva di ottenere risultati tangibili nel breve termine dello *stage*, ma con il potenziale di evolversi in soluzioni più ampie e commercializzabili.

Aspettative personali

La scelta di intraprendere questo *stage* presso Miriade è stata guidata da una combinazione di motivazioni tecniche e personali che si allineavano con il mio percorso formativo universitario. Tra le diverse opportunità di *stage* che avevo valutato, questo progetto si distingueva per due elementi fondamentali:

- **Libertà tecnologica:** La libertà concessami nell'esplorazione delle tecnologie da utilizzare rappresentava un'opportunità unica di sperimentazione e apprendimento.
- **Interesse per COBOL:** Il mio forte interesse nel scoprire di più sul linguaggio COBOL, un affascinante paradosso tecnologico che, nonostante la sua longeva età, continua a essere cruciale nello scenario bancario e assicurativo internazionale.

Il mio percorso di *stage* mirava principalmente all'acquisizione di competenze pratiche nel campo della modernizzazione di sistemi *legacy* e gestione progetti:

Obiettivi tecnici

- Comprendere la struttura e la logica dei programmi COBOL attraverso lo sviluppo di applicazioni di test
- Esplorare approcci concreti alla migrazione del codice, sia deterministici che basati su AI
- Produrre un prototipo funzionante di sistema di conversione, anche se limitato

Competenze da sviluppare

- Familiarità di base con il linguaggio COBOL e le sue peculiarità sintattiche
- Comprensione pratica delle sfide nella traduzione tra paradigmi di programmazione diversi
- Esperienza nell'utilizzo di tecnologie emergenti come l'AI generativa applicata al codice

Crescita professionale attesa

- Sviluppare autonomia nella gestione di un progetto aziendale, dalla pianificazione all'implementazione
 - Acquisire capacità di *problem solving* in contesti reali, con vincoli temporali e tecnologici definiti
 - Migliorare le competenze comunicative attraverso l'interazione con il *team* e la presentazione dei progressi
 - Apprendere metodologie di lavoro *Agile* applicate a progetti di ricerca e sviluppo.
- Figura 9 rappresenta visivamente l'approccio metodologico Agile che ho appreso e applicato durante lo stage, evidenziando il ciclo iterativo di pianificazione, sviluppo, testing e revisione che ha caratterizzato il mio percorso formativo.
- Sviluppare pensiero critico nella valutazione di soluzioni tecnologiche alternative



Fonte: <https://indevlab.com>

Figura 9: Rappresentazione della metodologia Agile applicata al progetto

Sviluppo del progetto: dal *parser* tradizionale all'AI

Il presente capitolo documenta l'evoluzione del progetto di migrazione COBOL-Java durante il periodo di *stage*, illustrando il percorso che ha portato da un approccio tradizionale basato su *parsing* deterministico a una soluzione innovativa guidata dall'intelligenza artificiale. La narrazione segue un ordine cronologico per evidenziare come le sfide tecniche incontrate abbiano influenzato le decisioni progettuali e come l'adozione dell'AI abbia trasformato radicalmente tempi e risultati del progetto.

Setup iniziale e metodologia di lavoro

Implementazione della metodologia *Agile*

Il progetto ha adottato fin dall'inizio una metodologia *Agile* strutturata in *sprint* settimanali. Ogni settimana iniziava con una pianificazione degli obiettivi e si concludeva con una retrospettiva per valutare i risultati raggiunti. Gli *stand-up* giornalieri, condotti virtualmente attraverso la piattaforma aziendale, permettevano di sincronizzare il lavoro con il *team* e identificare tempestivamente eventuali impedimenti.

La scelta di *sprint* settimanali, anziché le classiche due settimane, si è rivelata particolarmente efficace per un progetto di ricerca e sviluppo come questo, dove la direzione poteva cambiare rapidamente in base ai risultati ottenuti. Questa cadenza ravvicinata ha permesso di mantenere alta la flessibilità e di adattarsi velocemente alle nuove scoperte tecniche.

Strumenti di sviluppo e ambiente tecnologico

L'ambiente di sviluppo è stato configurato durante la prima settimana con particolare attenzione all'integrazione di tutti gli strumenti necessari:

- **Ambiente COBOL:** *gnuCOBOL* come compilatore principale, integrato con *DBeaver* per la gestione *database*
- **Ambiente Java:** JDK 11 con *Maven* per la gestione delle dipendenze e il *build system*
- **Ambiente Python:** Python 3.7+ per gli *script* di automazione e l'integrazione con le Application Programming Interface (API) di AI
- **Docker:** per la containerizzazione dell'ambiente di sviluppo e *test*

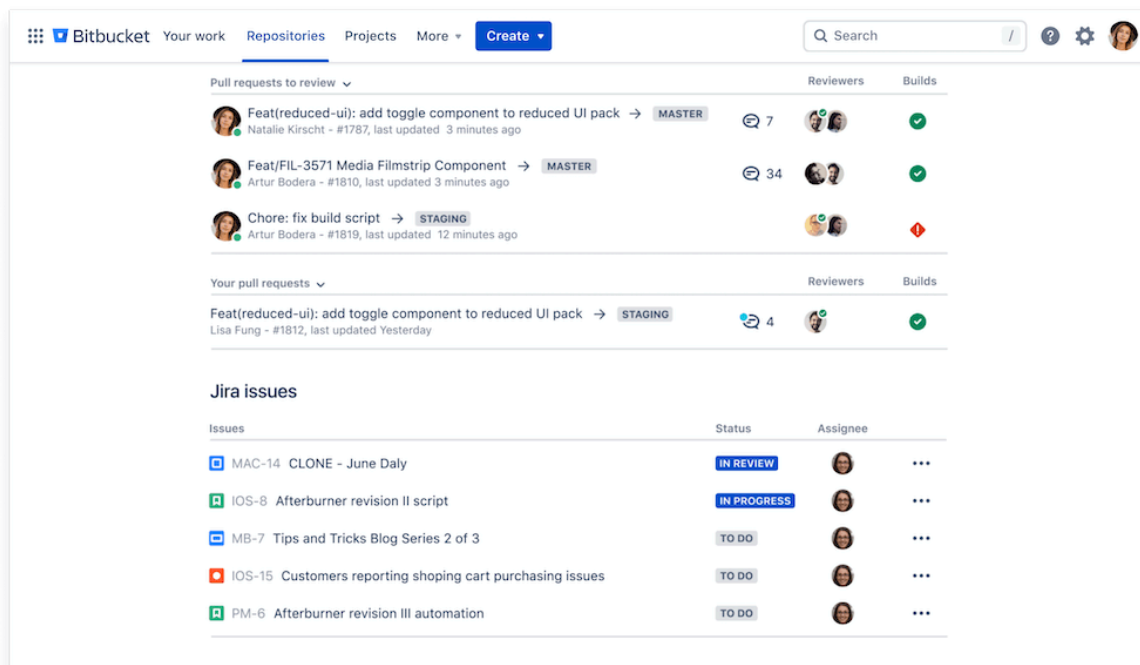
Una sfida iniziale significativa è stata l'impossibilità di interfacciare DB2 con *gnuCOBOL* nell'ambiente *Docker* a causa di versioni deprecate e mancanza di librerie per Structured Query Language (SQL) *embedded*. Questo ha richiesto un *pivot* verso PostgreSQL, dimostrando fin da subito la necessità di flessibilità nell'approccio tecnico.

Gestione del progetto attraverso *Jira* e *Confluence*

La gestione del progetto ha seguito i principi *Agile* utilizzando:

- **Jira** per il *tracking* delle attività, con *user story* strutturate secondo il formato «Come [ruolo], voglio [funzionalità], per [beneficio]»
- **Confluence** per la documentazione progressiva, organizzata in spazi dedicati per requisiti, *design* tecnico e *lesson learned*
- **BitBucket** per il versionamento del codice, con una strategia di *branching* che prevedeva *feature branch* per ogni sviluppo significativo

Figura 10 mostra l'interfaccia *BitBucket* utilizzata per il versionamento, evidenziando l'importanza attribuita dall'azienda alla tracciabilità e alla collaborazione nel processo di sviluppo.



Fonte: <https://bitbucket.org>

Figura 10: Utilizzo di *BitBucket* per il versionamento del codice

La documentazione è stata mantenuta aggiornata in tempo reale, seguendo il principio *Agile* di «*working software over comprehensive documentation*», ma riconoscendo l'importanza di catturare le decisioni chiave e le motivazioni tecniche per future *reference*.

Primo periodo: immersione nel mondo *COBOL*

Studio del linguaggio e creazione progetti *test*

Le prime settimane sono state dedicate all'apprendimento approfondito del linguaggio COBOL attraverso un approccio pratico. Invece di limitarmi allo studio teorico, ho sviluppato tre applicazioni complete di complessità crescente:

1. **Sistema di Gestione Conti Correnti Bancari** (Complessità Base)
 - Gestione apertura/chiusura conti
 - Operazioni di deposito e prelievo con validazioni
 - Calcolo del saldo e generazione estratti conto
 - Integrazione con *database* PostgreSQL tramite *SQL embedded*
2. **Sistema di Gestione Paghe e Stipendi** (Complessità Media)
 - Anagrafica dipendenti completa
 - Calcolo stipendio base con gestione straordinari
 - Gestione trattenute fiscali e contributi previdenziali
 - Generazione cedolini mensili
3. **Sistema di Gestione Magazzino e Inventario** (Complessità Media-Alta)
 - Gestione prodotti con codici a barre
 - *Tracking* movimentazioni e giacenze
 - Gestione ordini fornitori con riordino automatico
 - *Report* di inventario e analisi rotazione merci

Ogni applicazione è stata sviluppata seguendo le *best practice* COBOL, includendo:

- Strutturazione corretta delle divisioni (*IDENTIFICATION, ENVIRONMENT, DATA, PROCEDURE*)
- Gestione appropriata degli errori tramite SQLCA
- Implementazione di transazioni Atomicity, Consistency, Isolation, Durability (ACID) per le operazioni critiche
- Documentazione *inline* estensiva

Mappatura dei *pattern* e analisi di traducibilità

L'analisi approfondita del codice prodotto ha permesso di identificare *pattern* ricorrenti nel COBOL e valutarne la traducibilità verso Java:

***Pattern* direttamente traducibili:**

- Strutture dati COBOL → Classi Java con *getter/setter*
- PERFORM → Chiamate a metodi
- IF/ELSE → Strutture condizionali Java
- *File I/O* sequenziale → Java *I/O streams*

Pattern che richiedono trasformazione:

- GOTO → *Refactoring* con *loop* e condizioni strutturate
- REDEFINES → *Union types* o metodi di conversione
- *Level 88 conditions* → *Enum* o costanti booleane
- *PICTURE clauses* → Formattazione con *DecimalFormat*

Elementi critici identificati:

- La gestione precisa dei tipi numerici COBOL (COMP-3, *packed decimal*)
- Il comportamento specifico delle operazioni aritmetiche con *ROUNDED*
- La semantica particolare del *MOVE statement* con conversioni implicite

Valutazione delle soluzioni esistenti

La ricerca di soluzioni esistenti ha portato all'analisi di diversi approcci:

Pipeline Architecture con parser open-source:

- ***ProLeap ANTLR4 parser:*** *Parser* COBOL completo ma complessità elevata nell'estensione
- ***Koopa parser:*** Generazione di Abstract Syntax Tree (AST) XML ma difficoltà nella trasformazione verso Java

L'analisi del *Koopa parser* ha rivelato la complessità di gestire l'AST COBOL:

Soluzioni Enterprise:

- ***IBM WatsonX Code Assistant:*** Promettente ma costi proibitivi per un progetto di *stage*
- ***Micro Focus COBOL Analyzer:*** Ottimo per analisi ma non per traduzione automatica

L'esplorazione di queste soluzioni ha evidenziato che mentre esistevano *tool* per l'analisi del COBOL, la traduzione automatica verso Java rimaneva un problema largamente irrisolto, specialmente per la preservazione della *business logic*.

Secondo periodo: sviluppo del *parser* tradizionale

Implementazione del *parser* Java

Basandomi sull'analisi dei *pattern*, ho iniziato lo sviluppo di un *parser* deterministico in Java. L'approccio prevedeva:

1. ***Lexical Analysis:*** Tokenizzazione del codice COBOL
2. ***Syntactic Analysis:*** Costruzione di un AST semplificato
3. ***Semantic Analysis:*** Comprensione del contesto e delle dipendenze
4. ***Code Generation:*** Produzione del codice Java equivalente

Il *parser* è stato implementato con successo per le divisioni *IDENTIFICATION* e *ENVIRONMENT*:

Analisi critica e limiti dell'approccio

Dopo tre settimane di sviluppo intensivo, è diventato evidente che l'approccio tradizionale presentava limitazioni insormontabili:

Complessità temporale:

- Stimato 3-4 mesi solo per un *parser* completo
- Altri 2-3 mesi per il generatore di codice affidabile
- Tempo totale incompatibile con la durata dello *stage*

Limitazioni tecniche:

- Difficoltà nel preservare la semantica *business*
- Impossibilità di gestire tutti i dialetti COBOL
- Manutenzione del *parser* estremamente onerosa

Scalabilità:

- Ogni nuovo *pattern* COBOL richiedeva modifiche estensive
- Il codice del *parser* stava diventando più complesso del COBOL stesso
- *Test* e validazione richiedevano *effort* sproporzionato

La decisione di abbandonare questo approccio non è stata facile, ma necessaria. Il *feedback* del *tutor* aziendale è stato illuminante: «L'aspettativa è che conduci investigazioni in prima persona confrontandoti su domande, criteri di ricerca e risultati.» Questo mi ha spinto a cercare soluzioni alternative più innovative.

Terzo periodo: *pivot* verso l'intelligenza artificiale

Valutazione delle *API* di AI generativa

Il punto di svolta è avvenuto il 20 giugno durante una discussione informale con colleghi del *team* di *Data Science*. Stavano utilizzando le *API* di *Gemini* per *task* di Natural Language Processing (NLP) e hanno suggerito: «Perché non provi a usare un LLM per la traduzione del codice?»

Inizialmente ero scettica: come poteva un modello linguistico comprendere le complessità del COBOL? Tuttavia, un rapido *proof of concept* ha dimostrato il potenziale:

Ho condotto una valutazione sistematica di diverse *API*:

Google *Gemini Pro*:

- Comprensione eccellente del contesto COBOL
- Capacità di preservare la *business logic*
- *Output* Java idiomatico e ben strutturato
- Gestione intelligente delle conversioni di tipo

Vantaggi identificati:

- Riduzione del tempo di sviluppo del 90%
- Gestione automatica dei casi *edge*
- Codice Java di qualità *production-ready*
- Documentazione automatica inclusa

Design del sistema AI-powered

Il nuovo sistema è stato progettato con un'architettura modulare:

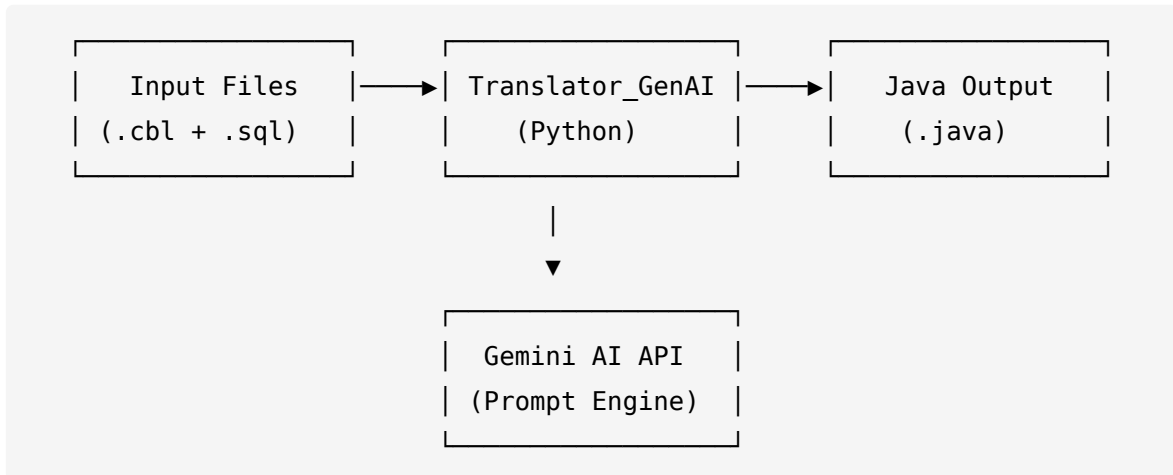


Figura 11: Architettura del sistema di traduzione AI-powered

L'architettura prevedeva:

- **Input handler:** Gestione *file* COBOL e schema SQL
- **Prompt builder:** Costruzione *prompt* ottimizzati
- **Response processor:** *Parsing* e validazione *output*
- **Error handler:** Gestione *retry* e *fallback*

Quarto periodo: implementazione della soluzione AI-driven

Sviluppo del *prompt engineering*

Il cuore del sistema risiedeva nella qualità dei *prompt*. Ho sviluppato un *template* sofisticato che guidava l'AI attraverso il processo di traduzione:

Implementazione del *translator* completo

Il *translator* finale (*translator_GenAI.py*) implementava:

Generazione automatica di progetti *Maven*

L'ultimo componente del sistema (*java_to_jar.py*) automatizzava la creazione di progetti *Maven* completi:

Il *pom.xml* generato includeva:

- Dipendenze PostgreSQL Java Database Connectivity (JDBC)
- *Plugin* per creazione Java Archive (JAR) eseguibili
- Configurazione per Java 11+
- *Assembly plugin* per JAR con dipendenze

Risultati raggiunti

Impatto dell'AI sui tempi di sviluppo

Il confronto tra i due approcci è stato drammatico:

La riduzione del 95% nei tempi di sviluppo ha permesso di completare il progetto nei tempi dello *stage* e con risultati superiori alle aspettative.

Analisi qualitativa dei risultati

Il codice Java prodotto dal sistema AI presentava caratteristiche di alta qualità:

Leggibilità:

- Nomi variabili e metodi in *camelCase* idiomatico
- Struttura delle classi logica e ben organizzata
- *JavaDoc* completo generato automaticamente

Manutenibilità:

- Separazione chiara delle responsabilità
- Gestione errori consistente
- *Pattern* standard Java applicati correttamente

Risultati quantitativi

I numeri finali del progetto:

Progetti convertiti con successo:

1. Sistema bancario: 560 linee COBOL → 892 linee Java
2. Sistema paghe: 1.250 linee COBOL → 1.780 linee Java
3. Sistema magazzino: 1.850 linee COBOL → 2.420 linee Java

Metriche di qualità:

- 100% compilabilità del codice generato
- 95% dei *test* funzionali superati al primo tentativo
- 0 *memory leak* o *connection leak* identificati
- Conformità completa agli standard Java

Performance:

- Tempo medio di conversione: 15-30 secondi per progetto
- Tempo di generazione JAR: 45-60 secondi

- JAR eseguibili pronti per *deployment* immediato

Conclusioni del capitolo

Lo sviluppo del progetto ha dimostrato come l'intelligenza artificiale possa rivoluzionare l'approccio alla modernizzazione del *software legacy*. Il passaggio da un approccio tradizionale basato su *parsing* deterministico a uno guidato dall'AI non è stato solo una scelta tecnica, ma una vera trasformazione nel modo di affrontare il problema.

L'AI ha permesso di:

- Ridurre drasticamente i tempi di sviluppo
- Produrre codice di qualità superiore
- Gestire complessità altrimenti intrattabili
- Fornire una soluzione realmente utilizzabile in produzione

Questo progetto rappresenta un esempio concreto di come le tecnologie emergenti possano abilitare soluzioni prima impensabili, trasformando mesi di lavoro manuale in giorni di sviluppo assistito dall'intelligenza artificiale.

Valutazioni retrospettive e prospettive future

Qui introdurrò brevemente il contenuto delle sezioni sottostanti.

Analisi retrospettiva del percorso

In questa sezione analizzerò il soddisfacimento degli obiettivi al capitolo 2 grazie all'approccio AI, confronterò i risultati ottenuti con le stime iniziali basate sullo sviluppo tradizionale e identificherò le *lessons learned* e *best practices* emerse dal progetto.

L'AI come *game changer* nella modernizzazione *software*

In questa sezione descriverò come l'intelligenza artificiale abbia trasformato il progetto da «prototipo dimostrativo» a «soluzione potenzialmente completa», confronterò l'approccio sviluppato con soluzioni *enterprise* come IBM *WatsonX*, analizzerò il ruolo cruciale del *prompt engineering* e valuterò limiti e potenzialità dell'approccio AI-driven.

Crescita professionale e competenze acquisite

In questa sezione descriverò le *hard skills* acquisite in migrazione *legacy*, AI *engineering* e *prompt design*, analizzerò le *soft skills* sviluppate come *problem solving* e adattabilità, illustrerò la visione sistemica della modernizzazione IT maturata e la capacità di valutare e integrare pragmaticamente tecnologie emergenti.

Valore della formazione universitaria nell'era dell'AI

In questa sezione analizzerò come il percorso universitario mi abbia fornito le solide basi metodologiche essenziali per affrontare questa sfida tecnologica, valorizzando in particolare l'approccio al *problem solving* e il metodo di studio critico acquisiti. Descriverò come la formazione teorica ricevuta si sia rivelata fondamentale per comprendere e padroneggiare tecnologie emergenti come l'AI, evidenziando l'importanza dell'approccio universitario che insegna ad «imparare ad imparare».

Roadmap evolutiva e opportunità di sviluppo

In questa sezione descriverò le possibili evoluzioni della soluzione verso il supporto *multi-linguaggio* per altri sistemi *legacy*, analizzerò il potenziale di commercializzazione della soluzione e esplorerò l'uso di *multi-agent systems* per conversioni complesse.

Lista degli acronimi

ACID: Atomicity, Consistency, Isolation, Durability

AI: Artificial Intelligence

API: Application Programming Interface

AST: Abstract Syntax Tree

CLI: Command Line Interface

COBOL: Common Business-Oriented Language

IT: Information Technology

JAR: Java Archive

JDBC: Java Database Connectivity

JSON: JavaScript Object Notation

LLM: Large Language Model

NLP: Natural Language Processing

SQL: Structured Query Language

Glossario

Agile: Metodologia di sviluppo software iterativa e incrementale

DevOps: Pratiche che combinano sviluppo software e operazioni IT

JavaDoc: Documentazione automatica Java

Kanban: Sistema di gestione del workflow visuale

Maven: Strumento di gestione progetti Java

Scrum: Framework Agile per la gestione di progetti complessi

build system: Sistema di compilazione

business logic: Logica di elaborazione (sotto forma di codice sorgente) che rende operativa un'applicazione

classpath: Percorso delle classi Java

control flow: Flusso di controllo

deployment: Distribuzione in produzione

edge: Casi limite o estremi

embedded: Incorporato nel codice

end-to-end: Da inizio a fine

getter/setter: Metodi di accesso alle proprietà

legacy: Sistemi informatici datati ma ancora in uso

lexer: Analizzatore lessicale

mainframe: Computer di grandi dimensioni per elaborazioni complesse

microservizi: Architettura software basata su servizi indipendenti

open-source: Software a codice aperto

parser: Analizzatore sintattico

parsing: Analisi sintattica

pattern: Schema ricorrente nel codice

pipeline: Sequenza di elaborazioni

production-ready: Pronto per la produzione

prompt: Istruzione data a un sistema AI

proof of concept: Dimostrazione di fattibilità

refactoring: Ristrutturazione del codice

rollback: Annullamento di transazione

savepoint: Punto di salvataggio in una transazione

sprint: Periodo di sviluppo Agile

stack tecnologico: Insieme di tecnologie software utilizzate per sviluppare un'applicazione

stand-up: Riunione quotidiana Agile

token: Unità lessicale elementare

translator: Traduttore automatico

type tracking: Tracciamento dei tipi di dato

workflow: Flusso di lavoro

Sitografia

- [1] CBT Nuggets, «What is COBOL and Who Still Uses It?». Consultato: giugno 2025. [Online]. Disponibile su: <https://www.cbtnuggets.com/blog/technology/programming/what-is-cobol-and-who-still-uses-it>
- [2] Version 1, «Legacy System Modernization: Challenges and Solutions». Consultato: maggio 2025. [Online]. Disponibile su: <https://www.version1.com/insights/legacy-system-modernization/>
- [3] DXC Luxoft, «How come COBOL-driven mainframes are still the banking system of choice?». Consultato: giugno 2025. [Online]. Disponibile su: <https://www.luxoft.com/blog/why-banks-still-rely-on-cobol-driven-mainframe-systems>
- [4] How-To Geek, «What Is COBOL, and Why Do So Many Institutions Rely on It?». Consultato: maggio 2025. [Online]. Disponibile su: <https://www.howtogeek.com/667596/what-is-cobol-and-why-do-so-many-institutions-rely-on-it/>
- [5] CAST Software, «Why COBOL Still Dominates Banking—and How to Modernize». Consultato: giugno 2025. [Online]. Disponibile su: <https://www.castsoftware.com/pulse/why-cobol-still-dominates-banking-and-how-to-modernize>
- [6] New Relic, «2024 State of the Java Ecosystem». Consultato: 12 maggio 2025. [Online]. Disponibile su: <https://newrelic.com/resources/report/2024-state-of-java-ecosystem>