AM - ANALISI DI MANUTENZIONE

CodeFace4Smells

Riferimento	
Versione	1.0
Data	07/07/2025
Destinatario	Prof. Andrea De Lucia
Presentato da	Antonio Ferrentino, Francesco Perilli, Giuseppe Napolitano

Composizione gruppo	
Antonio Ferrentino	0522501898
Francesco Perilli	0522502073
Giuseppe Napolitano	0522501961

Cronologia revisioni

Data	Versione	Descrizione	Autori
07/07/2025	0.1	Inizio stesura del documento	Antonio Ferrentino
08/07/2025	0.2	Suddivisione in capitoli del documento	Francesco Perilli
08/07/2025	0.3	Stesura della sezione relativa al database	Antonio Ferrentino
09/07/2025	0.4	Stesura iniziale della sezione 3	Francesco Perilli
12/07/2025	0.5	Completamento sezione 3	Francesco Perilli
12/07/2025	0.6	Stesura e completamento sezione 4	Antonio Ferrentino
15/07/2025	0.7	Stesura e completamento sezione 5	Francesco Perilli
16/07/2025	1.0	Revisione documento	Giuseppe Napolitano, Antonio Ferrentino, Francesco Perilli

Contents

1 Scopo del documento

Lo scopo del presente documento è illustrare la modalità con cui è stata svolta l'attività di manutenzione sul software **CodeFace4Smells**. In particolare, sarà presentata un'analisi degli impatti (*impact analysis*) volta a identificare le componenti del sistema maggiormente soggette a cambiamenti, applicando criteri strutturali e storici.

Il progetto ha previsto il tracciamento delle modifiche e la successiva analisi delle dipendenze interne al repository, con l'obiettivo di comprendere le aree più critiche e proporre una strategia di manutenzione efficace.

2 Panoramica del sistema attuale

Il progetto **CodeFace4Smells** si basa sullo strumento open source *CodeFace*, utilizzato per l'analisi evolutiva di progetti software. Il sistema originario, pur essendo funzionalmente ricco, non presentava una documentazione, ma solo la presenza di 3 *README*, tale da rendere difficile la comprensione globale della sua architettura e delle sue funzionalità principali.

Per questo motivo si è reso necessario intraprendere un'attività preliminare di *reverse engineering* per ricostruire, a partire dal codice sorgente e dalle evidenze storiche, una descrizione funzionale e strutturale coerente con lo stato attuale del software.

2.1 Reverse engineering

Inizialmente, **CodeFace4Smells** si presentava privo di una documentazione. È stata quindi condotta un'attività di *reverse engineering* con l'obiettivo di ricostruire i requisiti funzionali e mappare le relazioni tra le componenti del sistema.

L'approccio seguito è stato strutturato secondo il paradigma **goals/models/tools**, articolato come segue:

- **Goal:** l'obiettivo dell'attività è stato quello di comprendere il sistema ed ottenere una documentazione ad alto livello, focalizzata sui requisiti funzionali e sulle relazioni tra gli artefatti software;
- **Models:** sono stati prodotti due modelli principali:
  - una lista strutturata dei requisiti funzionali con relativa descrizione;
  - una matrice di tracciabilità per mantenere le relazioni forward e backward tra requisiti e codice.
- **Tools:** gli strumenti utilizzati includono:
  - le funzionalità *Find in Files* e *Find Usages* di PyCharm, basate su tecniche di analisi sintattica e lessicale, per individuare riferimenti e utilizzi di elementi specifici all'interno del codice.

2.2 Descrizione del sistema attuale

CodeFace4Smells è una piattaforma di analisi automatizzata orientata allo studio dell'evoluzione del software e alla rilevazione dei *code smells* nei repository di progetti open source. Si tratta di un'estensione e riadattamento dello strumento *CodeFace*, progettato per supportare attività di manutenzione e refactoring basate su dati storici e strutturali.

Il sistema consente di:

- raccogliere e correlare informazioni da repository Git, issue tracker e sistemi di tagging;
- generare matrici di co-evoluzione tra moduli e sviluppatori;
- individuare *code smells* attraverso metriche e regole definite a partire dal grafo delle dipendenze interne;
- produrre report PDF e visualizzazioni grafiche delle dipendenze e dei cluster.

La piattaforma è pensata per agevolare lo studio dell'evoluzione software da parte di ricercatori e manutentori, consentendo un'analisi approfondita delle modifiche apportate nel tempo e delle aree soggette a degrado strutturale.

Per promuovere la collaborazione e la replicabilità, **CodeFace4Smells** fornisce anche una serie di script R e Python, che permettono l'automazione delle analisi e la personalizzazione dei processi, rendendo la piattaforma flessibile e adattabile a diversi contesti software.

2.3 Attori del sistema

Gli attori coinvolti nel sistema **CodeFace4Smells** sono i seguenti:

- **Utente ospite:** rappresenta un utilizzatore che può consultare la documentazione generale del sistema, accedere al repository pubblico del progetto e visualizzare esempi di analisi già eseguite;
- **Manutentore software:** è l'utente primario del sistema. Ha le seguenti responsabilità:
  - importare ed analizzare nuovi repository Git;
  - consultare le metriche software, i grafi di dipendenza e i code smells rilevati;
  - eseguire confronti tra versioni successive per identificare modifiche strutturali;
  - generare report PDF e CSV personalizzati;
  - personalizzare le soglie delle metriche e i criteri di rilevamento degli smells;
- **Ricercatore:** utente avanzato che può:
  - configurare analisi batch su più progetti software;
  - esportare i dati per analisi esterne (es. R, Python, strumenti statistici);
  - sviluppare e testare nuove euristiche di clustering o tracciamento evolutivo;

2.4 Requisiti funzionali

Gestione Sistema

RF_CF_1 – Installazione automatica: il sistema dovrà permettere l'installazione automatica tramite Vagrant e provisioning della macchina virtuale. – Priorità Alta
RF_CF_2 – Setup ambientale: il sistema dovrà permettere l'inizializzazione di un ambiente Python virtuale con tutte le dipendenze necessarie. – Priorità Alta
RF_CF_3 – Configurazione MySQL: il sistema dovrà permettere la configurazione automatica e manuale di database MySQL, incluso l'inserimento dello schema. – Priorità Alta

Gestione Analisi Progetto

RF_CF_4 – Importazione repository: il sistema dovrà permettere l'importazione di un repository Git contenente il codice sorgente da analizzare. – Priorità Alta
RF_CF_5 – Esecuzione analisi: il sistema dovrà permettere l'esecuzione dell'analisi tramite il comando `codeface run`, sulla base di un file `.conf` dedicato al progetto. – Priorità Alta
RF_CF_6 – Rilevamento revisioni e tag: il sistema dovrà rilevare automaticamente le revisioni e i tag del repository specificato. – Priorità Alta
RF_CF_7 – Analisi temporale: il sistema dovrà supportare l'analisi delle evoluzioni temporali del progetto tramite script R dedicati. – Priorità Alta
RF_CF_8 – Analisi strutturale: il sistema dovrà generare grafi di dipendenza e clusterizzazione degli sviluppatori a partire dai commit. – Priorità Alta
RF_CF_9 – Rilevamento code smells: il sistema dovrà individuare code smells utilizzando criteri strutturali, metrici e storici. – Priorità Alta

Gestione Output e Reporting

RF_CF_10 – Generazione report: il sistema dovrà generare report PDF e HTML contenenti risultati di clustering, metriche e visualizzazioni. – Priorità Alta
RF_CF_11 – Visualizzazione grafi: il sistema dovrà generare e visualizzare grafi di chiamata e co-evoluzione tra moduli. – Priorità Media
RF_CF_12 – Esportazione risultati: il sistema dovrà salvare i risultati dell'analisi in directory configurabili, in formato CSV o SQLite. – Priorità Alta

Interfaccia Web

RF_CF_13 – Accesso interfaccia Shiny: il sistema dovrà fornire un'interfaccia web Shiny per la consultazione dei risultati in locale. – Priorità Media
RF_CF_14 – Avvio server interattivo: il sistema dovrà supportare l'avvio manuale o automatizzato del server Shiny tramite script. – Priorità Media

Testing e Manutenzione

RF_CF_15 – Esecuzione test: il sistema dovrà supportare l'esecuzione di test di integrazione tramite `codeface test -c conf`. – Priorità Alta
RF_CF_16 – Gestione ambienti separati: il sistema dovrà supportare l'utilizzo di un database di test separato da quello di produzione. – Priorità Alta
RF_CF_17 – Logging: il sistema dovrà produrre file di log dettagliati per ogni operazione eseguita. – Priorità Alta

Categoria	Identificativo	Descrizione	Priorità
Gestione Sistema	RF_CF_1	Installazione automatica	Alta
	RF_CF_2	Setup ambientale	Alta
	RF_CF_3	Configurazione MySQL	Alta
Gestione Analisi Progetto	RF_CF_4	Importazione repository	Alta
	RF_CF_5	Esecuzione analisi	Alta
	RF_CF_6	Rilevamento revisioni e tag	Alta
	RF_CF_7	Analisi temporale	Alta
	RF_CF_8	Analisi strutturale	Alta
	RF_CF_9	Rilevamento code smells	Alta
Gestione Output e Reporting	RF_CF_10	Generazione report	Alta
	RF_CF_11	Visualizzazione grafi	Media
	RF_CF_12	Esportazione risultati	Alta
Interfaccia Web	RF_CF_13	Accesso interfaccia Shiny	Media
	RF_CF_14	Avvio server interattivo	Media
Testing e Manutenzione	RF_CF_15	Esecuzione test	Alta
	RF_CF_16	Gestione ambienti separati	Alta
	RF_CF_17	Logging	Alta

2.5 Architettura del sistema

L'architettura del sistema **CodeFace4Smells** è organizzata secondo un paradigma a **pipeline modulare orchestrata**, in cui componenti eterogenei (script Python, script R, strumenti di visualizzazione) vengono eseguiti in sequenza tramite un orchestratore centrale.

I dati vengono analizzati in modalità batch e i risultati sono visualizzabili a posteriori tramite report statici o dashboard locali.

L'architettura è suddivisa logicamente nei seguenti livelli:

- **Orchestrazione:** componenti come `cli.py`, `project.py` e gli script di provisioning coordinano l'intero processo.
- **Elaborazione:** script Python e R per l'analisi strutturale, temporale e rilevamento di code smells.
- **Output:** generazione di report PDF/HTML e visualizzazioni interattive tramite Shiny, su dati preelaborati.

Questa architettura rende il sistema flessibile, estensibile e particolarmente adatto ad attività di analisi automatizzata di progetti software su larga scala.

2.6 Database

2.6.1 Modello EER

Per visualizzare il modello EER, ecco il file: MODELLO EER V.1

2.6.2 Schema relazionale

Project (id, name, analysisMethod, analysisTime)
Person (id, name, email1, email2*, email3*, email4*, email5*, *projectId*↑)
Issue (id, bugId, creationDate, modifiedDate*, url*, isRegression*, status, resolution*, priority, severity, *createdBy*↑, assignedTo↑, *projectId*↑, subComponent, subSubComponent*, version*)
Issue_Comment (id, *who*↑, *fk_issueId*↑, commentDate*)
Issue_History (id, changeDate, field, oldValue*, newValue*, *who*↑, *issueId*↑)
Release_Timeline (id, type, tag, date*, *projectId*↑)
Release_Range (id, *releaseStartId*↑, *releaseEndId*↑, *projectId*↑, releaseRCStartId↑)
Commit (id, commitHash, commitDate, authorDate, authorTimeOffset, authorTimezones*, ChangedFiles*, AddedLines*, DeletedLines*, DiffSize*, CmtMsgLines*, CmtMsgBytes*, NumSignedOffs*, NumTags*, general*, TotalSubsys*, Subsys*, inRC*, AuthorSubsysSimilarity*, AuthorTaggersSimilarity*, TaggersSubsysSimilarity*, description*, corrective*, *author*↑, *projectId*↑, releaseRangeId↑)
Commit_Dependency (id, file, entityId, entityType, size, impl*, *commitId*↑)
Author_Commit_Stats (id, added*, deleted*, total*, numcommits*, *authorId*↑, *releaseRangeId*↑)
Commit_Communication (id, communicationType, *commitId*↑, *who*↑)
Mailing_List (id, name, description*, *projectId*↑)
Mail_Thread (id, subject*, creationDate*, numberOfAuthors, numberOfMessages, mailThreadId, *createdBy*↑, *projectId*↑, *releaseRangeId*↑, *mlId*↑)
Thread_Responses (*who*↑, *mailThreadId*↑, mailDate)
Mail (id, subject*, creationDate, *projectId*↑, *mlId*↑, *threadId*↑, *author*↑)
CC_List (*issueId*↑, *who*↑)
Issue_Duplicates (id, *originalBugId*↑, *duplicateBugId*↑)
Issue_Dependencies (id, *originalIssueId*↑, *dependentIssueId*↑)
Cluster (id, clusterNumber*, clusterMethod*, dot*, svg*, *projectId*↑, *releaseRangeId*↑)
Cluster_User_Mapping (id, *personId*↑, *clusterId*↑)
Edgelist (*clusterId*↑, *fromId*↑, *toId*↑, weight)
Plot_Bin (*plotID*↑, type, data)
Plots (id, name, labelx*, labely*, *projectId*↑, releaseRangeId↑)
Timeseries (*plotId*↑, time, value, value_scaled)
Sloccount_TS (*plotId*↑, time, person_months, total_cost, schedule_months, avg_devel)
Understand_Raw (*plotId*↑, time, kind, name*, variable, value)
PageRank (id, technique, name*, *releaseRangeId*↑)
PageRank_Matrix (*pageRankId*↑, *personId*↑, rankValue)
Per_Cluster_Statistics (*projectId*↑, *releaseRangeId*↑, *clusterId*↑, technique, num_members, added, deleted, total, numcommits, prank_avg)
Per_Person_Cluster_Statistics_View (projectId, releaseRangeId, clusterId, personId, technique, rankValue, added, deleted, total, numcommits)
URL_Info (id, type, url, *projectId*↑)
Thread_Density (id, num, density, type, *projectId*↑)
Freq_Subjects (id, subject, count, *projectId*↑, *releaseRangeId*↑, *mlId*↑)
TwoMode_Edgelist (*releaseRangeId*↑, source, *mlId*↑, *fromVert*↑, toVert, weight)
TwoMode_Vertices (*releaseRangeId*↑, source, *mlId*↑, name, degree, type)
Initiate_Response (*releaseRangeId*↑, *mlId*↑, *personId*↑, source, responses*, initiations*, responses_received*, deg*)

2.7 Testing

Prima di procedere con qualunque intervento di manutenzione, è stata condotta una sessione di **testing di sistema** su *CodeFace4Smells*, allo scopo di valutare il comportamento del sistema nella sua versione originale.

L'attività di testing non ha previsto l'esecuzione di test unitari o di integrazione automatizzati, ma si è basata esclusivamente su test manuali di sistema, eseguiti tramite l'interfaccia a riga di comando. Durante ciascun test, sono stati osservati con attenzione i messaggi di errore restituiti dal terminale Python e dall'ambiente R, con l'obiettivo di identificare comportamenti anomali o malfunzionamenti strutturali.

Ogni esecuzione è stata documentata annotando:

- lo **scenario di test** (es. comando utilizzato, progetto analizzato, configurazione fornita);
- l'**output del terminale** e gli eventuali errori o warning;
- le **azioni correttive** o modifiche necessarie per ottenere un funzionamento corretto.

Il testing è stato iterativo: ogni modifica al codice o alla configurazione è stata seguita da una nuova esecuzione, in un processo continuo di regressione manuale. Questo approccio ha permesso di monitorare con precisione l'impatto delle attività di manutenzione e garantire il mantenimento della funzionalità del sistema.

Una descrizione più dettagliata degli scenari testati e delle anomalie riscontrate è disponibile all'interno del **Piano di Test**.

3 Debug e manutenzione avanzata

Il progetto *Codeface* nasce come strumento avanzato per l'analisi di repository software open source, offrendo supporto per il tracciamento delle attività di sviluppo, l'estrazione di metriche temporali e la visualizzazione di dinamiche sociali tra i collaboratori. Tuttavia, l'architettura attuale presenta limiti strutturali e funzionali che ne compromettono la manutenibilità, la compatibilità e la stabilità su ambienti di lavoro moderni. Il presente documento propone formalmente una *Change Request (CR)* per avviare un'attività di refactoring profondo del codice esistente.

Project Name	Codeface4Smells
Project Manager	Antonio Ferrentino, Francesco Perilli, Giuseppe Napolitano
Requested by	Andrea De Lucia
Submitted by	Antonio Ferrentino, Francesco Perilli, Giuseppe Napolitano
Date Requested	07/05/2025
Evaluated by	Antonio Ferrentino, Francesco Perilli, Giuseppe Napolitano
Date Evaluated	17/07/2025
Decided by	Antonio Ferrentino, Francesco Perilli, Giuseppe Napolitano
Date Decided	17/07/2025

Change Request Identification	
Change Request #	CR_01
Change Title	Refactoring totale
Maintenance type	Manutenzione correttiva

3.1 Motivazione della CR

Nel corso delle attività di sviluppo e manutenzione del progetto, sono emerse criticità significative che giustificano la necessità di un refactoring completo:

- **Deprecazione delle librerie R**: diverse librerie fondamentali per il funzionamento di Codeface (es. `RMySQL`, `plyr`, `lubridate`) sono obsolete o incompatibili con le versioni correnti del linguaggio R e dei suoi pacchetti.
- **Accoppiamento rigido e codice non modulare**: l'attuale struttura del codice presenta forte accoppiamento tra i moduli di configurazione, database, analisi e visualizzazione, rendendo difficile l'estensione o la sostituzione di singoli componenti.
- **Errori sistemici e comportamenti inattesi**: il sistema produce errori critici in fasi basilari dell'esecuzione, come la generazione delle time series, la gestione delle boundaries o l'invocazione delle routine SLOCCount, dovuti a logiche fragili o gestione non robusta dei dati.
- **Difficoltà di debug e assenza di test**: la mancanza di un framework strutturato per la validazione del codice e la diagnosi degli errori rallenta i cicli di sviluppo e impedisce un corretto tracciamento delle regressioni.

3.2 Obiettivi del Refactoring

La CR ha come obiettivo principale la ristrutturazione dell'intero sistema, mantenendo la logica funzionale originale ma aggiornando profondamente l'implementazione. In particolare:

- **Sostituzione delle librerie deprecate** con alternative moderne, pienamente compatibili con le versioni attuali di R.
- **Modularizzazione del codice** mediante separazione delle responsabilità (es. configurazione, accesso ai dati, calcolo analitico, visualizzazione).
- **Introduzione di test automatici** (unitari e funzionali) per garantire affidabilità e prevenzione di regressioni.
- **Aggiunta di logging strutturato** e strumenti di debug per migliorare la tracciabilità degli errori.
- **Compatibilità con ambienti containerizzati**, come Docker, per facilitare lo sviluppo, il testing e il deployment.

3.3 Benefici Attesi

L'adozione di questa CR apporterà benefici significativi al progetto, tra cui:

- Maggiore stabilità e compatibilità del software;

- Facilità di manutenzione e onboarding di nuovi sviluppatori;
- Riduzione del debito tecnico e incremento dell'affidabilità;
- Migliore supporto per nuove funzionalità e progetti di ricerca.

3.4 Strategie di debugging

Le tecniche utilizzate per identificare e risolvere i problemi sono state:

- Inserimento di messaggi `print()` o `loginfo()` in punti strategici del codice R per verificare il contenuto degli oggetti `conf`, `boundaries`, `range.id`, `revisions`;
- Rimozione graduale di assegnazioni sospette, ad esempio l'istruzione:

  ```
  conf$pid <- old_pid
  ```

  che sovrascriveva un valore valido causando la perdita del `range.id` downstream;
- Analisi del contenuto del database tramite query dirette MySQL per validare la presenza di `release_range`, `release_timeline`, `commit`, `plots`, `sloccount_ts` e `cluster`.

4 Sistema Modificato

4.1 Descrizione del sistema attuale

Il sistema nel complesso non ha perso le sue funzionalità iniziali, tranne per il server e l'interfaccia **Shiny**, ormai non più disponibili. **CodeFace4Smells** è una piattaforma di analisi automatizzata orientata allo studio dell'evoluzione del software e alla rilevazione dei *code smells* nei repository di progetti open source. Si tratta di un'estensione e riadattamento dello strumento *CodeFace*, progettato per supportare attività di manutenzione e refactoring basate su dati storici e strutturali.

Il sistema consente di:

- raccogliere e correlare informazioni da repository Git, issue tracker e sistemi di tagging;
- generare matrici di co-evoluzione tra moduli e sviluppatori;
- individuare *code smells* attraverso metriche e regole definite a partire dal grafo delle dipendenze interne;
- produrre report PDF e visualizzazioni grafiche delle dipendenze e dei cluster.

La piattaforma è pensata per agevolare lo studio dell'evoluzione software da parte di ricercatori e manutentori, consentendo un'analisi approfondita delle modifiche apportate nel tempo e delle aree soggette a degrado strutturale.

Per promuovere la collaborazione e la replicabilità, **CodeFace4Smells** fornisce anche una serie di script R e Python, che permettono l'automazione delle analisi e la personalizzazione dei processi, rendendo la piattaforma flessibile e adattabile a diversi contesti software.

4.2 Attori del sistema

Gli attori coinvolti nel sistema **CodeFace4Smells** sono i seguenti:

- **Utente ospite:** rappresenta un utilizzatore che può consultare la documentazione generale del sistema, accedere al repository pubblico del progetto e visualizzare esempi di analisi già eseguite;
- **Manutentore software:** è l'utente primario del sistema. Ha le seguenti responsabilità:
  - importare ed analizzare nuovi repository Git;
  - consultare le metriche software, i grafi di dipendenza e i code smells rilevati;
  - eseguire confronti tra versioni successive per identificare modifiche strutturali;
  - generare report PDF e CSV personalizzati;
  - personalizzare le soglie delle metriche e i criteri di rilevamento degli smells;
- **Ricercatore:** utente avanzato che può:
  - configurare analisi batch su più progetti software;
  - esportare i dati per analisi esterne (es. R, Python, strumenti statistici);
  - sviluppare e testare nuove euristiche di clustering o tracciamento evolutivo;

4.3 Requisiti funzionali

Gestione Sistema

RF_CF_1 – Installazione automatica: il sistema dovrà permettere l'installazione automatica tramite Vagrant e provisioning della macchina virtuale. – Priorità Alta
RF_CF_2 – Setup ambientale: il sistema dovrà permettere l'inizializzazione di un ambiente Python virtuale con tutte le dipendenze necessarie. – Priorità Alta
RF_CF_3 – Configurazione MySQL: il sistema dovrà permettere la configurazione automatica e manuale di database MySQL, incluso l'inserimento dello schema. – Priorità Alta

Gestione Analisi Progetto

RF_CF_4 – Importazione repository: il sistema dovrà permettere l'importazione di un repository Git contenente il codice sorgente da analizzare. – Priorità Alta
RF_CF_5 – Esecuzione analisi: il sistema dovrà permettere l'esecuzione dell'analisi tramite il comando `codeface run`, sulla base di un file `.conf` dedicato al progetto. – Priorità Alta
RF_CF_6 – Rilevamento revisioni e tag: il sistema dovrà rilevare automaticamente le revisioni e i tag del repository specificato. – Priorità Alta
RF_CF_7 – Analisi temporale: il sistema dovrà supportare l'analisi delle evoluzioni temporali del progetto tramite script R dedicati. – Priorità Alta
RF_CF_8 – Analisi strutturale: il sistema dovrà generare grafi di dipendenza e clusterizzazione degli sviluppatori a partire dai commit. – Priorità Alta
RF_CF_9 – Rilevamento code smells: il sistema dovrà individuare code smells utilizzando criteri strutturali, metrici e storici. – Priorità Alta

Gestione Output e Reporting

RF_CF_10 – Generazione report: il sistema dovrà generare report PDF e HTML contenenti risultati di clustering, metriche e visualizzazioni. – Priorità Alta
RF_CF_11 – Visualizzazione grafi: il sistema dovrà generare e visualizzare grafi di chiamata e co-evoluzione tra moduli. – Priorità Media
RF_CF_12 – Esportazione risultati: il sistema dovrà salvare i risultati dell'analisi in directory configurabili, in formato CSV o SQLite. – Priorità Alta

Testing e Manutenzione

RF_CF_13 – Esecuzione test: il sistema dovrà supportare l'esecuzione di test di integrazione tramite `codeface test -c conf`. – Priorità Alta
RF_CF_14 – Gestione ambienti separati: il sistema dovrà supportare l'utilizzo di un database di test separato da quello di produzione. – Priorità Alta
RF_CF_15 – Logging: il sistema dovrà produrre file di log dettagliati per ogni operazione eseguita. – Priorità Alta

Categoria	Identificativo	Breve Descrizione	Priorità
Gestione Sistema	RF_CF_1	Installazione automatica	Alta
	RF_CF_2	Setup ambientale	Alta
	RF_CF_3	Configurazione MySQL	Alta
Gestione Analisi Progetto	RF_CF_4	Importazione repository	Alta
	RF_CF_5	Esecuzione analisi	Alta
	RF_CF_6	Rilevamento revisioni e tag	Alta
	RF_CF_7	Analisi temporale	Alta
	RF_CF_8	Analisi strutturale	Alta
	RF_CF_9	Rilevamento code smells	Alta
Gestione Output e Reporting	RF_CF_10	Generazione report	Alta
	RF_CF_11	Visualizzazione grafi	Media
	RF_CF_12	Esportazione risultati	Alta
Testing e Manutenzione	RF_CF_13	Esecuzione test	Alta
	RF_CF_14	Gestione ambienti separati	Alta
	RF_CF_15	Logging	Alta

4.4 Architettura del sistema

L'architettura del sistema **CodeFace4Smells** è organizzata secondo un paradigma a **pipeline modulare orchestrata**, in cui componenti eterogenei (script Python, script R, strumenti di visualizzazione) vengono eseguiti in sequenza tramite un orchestratore centrale.

I dati vengono analizzati in modalità batch e i risultati sono visualizzabili a posteriori tramite report statici o dashboard locali.

L'architettura è suddivisa logicamente nei seguenti livelli:

- **Orchestrazione:** componenti come `cli.py`, `project.py` e gli script di provisioning coordinano l'intero processo.
- **Elaborazione:** script Python e R per l'analisi strutturale, temporale e rilevamento di code smells.
- **Output:** generazione di report PDF/HTML e visualizzazioni interattive tramite Shiny, su dati preelaborati.

Questa architettura rende il sistema flessibile, estensibile e particolarmente adatto ad attività di analisi automatizzata di progetti software su larga scala.

4.5 Database

4.5.1 Modello EER

Per visualizzare il modello EER, ecco il file: MODELLO EER v.2

4.5.2 Schema relazionale

Project (id, name, analysisMethod, analysisTime)
Person (id, name, email1, email2*, email3*, email4*, email5*, *projectId*↑)
Issue (id, bugId, creationDate, modifiedDate*, url*, isRegression*, status, resolution*, priority,

severity, *createdBy*↑, assignedTo↑, *projectId*↑, subComponent, subSubComponent*, version*)
Issue_Comment (id, *who*↑, *fk_issueId*↑, commentDate*)
Issue_History (id, changeDate, field, oldValue*, newValue*, *who*↑, *issueId*↑)
Release_Timeline (id, type, tag, date*, *projectId*↑)
Release_Range (id, *releaseStartId*↑, *releaseEndId*↑, *projectId*↑, releaseRCStartId↑)
Commit (id, commitHash, commitDate, authorDate, authorTimeOffset, authorTimezones*, ChangedFiles*, AddedLines*, DeletedLines*, DiffSize*, CmtMsgLines*, CmtMsgBytes*, NumSignedOffs*, NumTags*, general*, TotalSubsys*, Subsys*, inRC*, AuthorSubsysSimilarity*, AuthorTaggersSimilarity*, TaggersSubsysSimilarity*, description*, corrective*, *author*↑, *projectId*↑, releaseRangeId↑)
Commit_Dependency (id, file, entityId, entityType, size, impl*, *commitId*↑)
Author_Commit_Stats (id, added*, deleted*, total*, numcommits*, *authorId*↑, *releaseRangeId*↑)
Commit_Communication (id, communicationType, *commitId*↑, *who*↑)
Mailing_List (id, name, description*, *projectId*↑)
Mail_Thread (id, subject*, creationDate*, numberOfAuthors, numberOfMessages, mailThreadId, *createdBy*↑, *projectId*↑, *releaseRangeId*↑, *mlId*↑)
Thread_Responses (*who*↑, *mailThreadId*↑, mailDate)
Mail (id, subject*, creationDate, *projectId*↑, *mlId*↑, *threadId*↑, *author*↑)
CC_List (*issueId*↑, *who*↑)
Issue_Duplicates (id, *originalBugId*↑, *duplicateBugId*↑)
Issue_Dependencies (id, *originalIssueId*↑, *dependentIssueId*↑)
Cluster (id, clusterNumber*, clusterMethod*, dot*, svg*, *projectId*↑, *releaseRangeId*↑, label*)
Cluster_User_Mapping (id, *personId*↑, *clusterId*↑)
Edgelist (*clusterId*↑, *fromId*↑, *toId*↑, weight)
Plot_Bin (*plotID*↑, type, data)
Plots (id, name, labelx*, labely*, *projectId*↑, releaseRangeId↑)
Timeseries (*plotId*↑, time, value, value_scaled)
Sloccount_TS (*plotId*↑, time, person_months, total_cost, schedule_months, avg_devel)
Understand_Raw (*plotId*↑, time, kind, name*, variable, value)
PageRank (id, technique, name*, *releaseRangeId*↑)
PageRank_Matrix (*pageRankId*↑, *personId*↑, rankValue)
Per_Cluster_Statistics (*projectId*↑, *releaseRangeId*↑, *clusterId*↑, technique, num_members, added, deleted, total, numcommits, prank_avg)
Per_Person_Cluster_Statistics_View (projectId, releaseRangeId, clusterId, personId, technique, rankValue, added, deleted, total, numcommits)
URL_Info (id, type, url, *projectId*↑)
Thread_Density (id, num, density, type, *projectId*↑)
Freq_Subjects (id, subject, count, *projectId*↑, *releaseRangeId*↑, *mlId*↑)
TwoMode_Edgelist (*releaseRangeId*↑, source, *mlId*↑, *fromVert*↑, toVert, weight)
TwoMode_Vertices (*releaseRangeId*↑, source, *mlId*↑, name, degree, type)
Initiate_Response (*releaseRangeId*↑, *mlId*↑, *personId*↑, source, responses*, initiations*, responses_received*, deg*)

5 Impact Analysis

L'analisi d'impatto ha l'obiettivo di valutare gli effetti delle modifiche software apportate ai moduli del sistema `CodeFace4Smells`, in termini di propagazione tra file e componenti. L'approccio seguito riprende il modello strutturato in **SIS**, **CIS**, **DIS**, **FPIS** e **AIS**.

5.1 Start Impact Set (SIS)

Lo *Start Impact Set (SIS)* include tutti i file direttamente modificati nei commit analizzati. I file identificati sono:

- `codeface/R/cluster/graph_comparison.r`
- `codeface/R/cluster/persons.r`
- `codeface/R/cluster/test_graph_comparison.r`
- `codeface/R/utils.r`
- `codeface/VCS.py`
- `codeface/configuration.py`
- `codeface/util.py`
- `CppStatsFilesUpdate/cppstats-0.8.4/analyses/featurelocations.py`
- `CppStatsFilesUpdate/cppstats-0.8.4/preparation.py`

5.2 Candidate Impact Set (CIS)

Il *Candidate Impact Set (CIS)* è stato definito considerando:

- i file che importano o dipendono direttamente dai moduli modificati;
- la struttura del progetto e le pipeline di elaborazione dati;
- i test e gli script di orchestrazione coinvolti.

Pertanto, il **CIS** è stato definito come:

- `codeface/project.py`
- `codeface/cli.py`
- `codeface/logger.py`
- `codeface/test/integration/test_exampleprojects.py`
- `codeface/dbmanager.py`
- `codeface/cluster/cluster.py`
- `codeface/commit_analysis.py`
- `integration-scripts/install_codeface_R.sh`
- `integration-scripts/provision.sh`

5.3 Discovered Impact Set (DIS)

Tramite analisi statica delle dipendenze e test eseguiti nel contesto progettuale, sono stati effettivamente rilevati come impattati (in esecuzione o tramite invocazione diretta) i seguenti file:

- `codeface/project.py`
- `codeface/cli.py`
- `codeface/cluster/cluster.py`
- `codeface/test/integration/test_exampleprojects.py`

5.4 False Positive Impact Set (FPIS)

All'interno del CIS sono presenti file che, nonostante siano stati considerati a rischio d'impatto, non risultano effettivamente coinvolti:

- `codeface/logger.py`
- `integration-scripts/provision.sh`
- `integration-scripts/install_codeface_R.sh`

5.5 Actual Impact Set (AIS)

L'*Actual Impact Set (AIS)* è definito come:

$$AIS = (SIS \cup DIS) - FPIS$$

Pertanto:
AIS = {

- `codeface/R/cluster/graph_comparison.r`,
- `codeface/R/cluster/persons.r`,
- `codeface/R/cluster/test_graph_comparison.r`,
- `codeface/R/utils.r`,
- `codeface/VCS.py`,
- `codeface/configuration.py`,
- `codeface/util.py`,
- `CppStatsFilesUpdate/cppstats-0.8.4/analyses/featurelocations.py`,
- `CppStatsFilesUpdate/cppstats-0.8.4/preparation.py`,
- `codeface/project.py`,
- `codeface/cli.py`,
- `codeface/cluster/cluster.py`,
- `codeface/test/integration/test_exampleprojects.py`

}

5.6 Metriche di valutazione

Per valutare l'efficacia dell'analisi d'impatto sono stati calcolati gli indicatori standard **precision** e **recall**, sulla base delle cardinalità dei set identificati:

Insieme	**Cardinalità**
SIS (Start Impact Set)	9
CIS (Candidate Impact Set)	9
DIS (Discovered Impact Set)	4
FPIS (False Positive Impact Set)	3
AIS (Actual Impact Set) = (SIS $\cup$ DIS) – FPIS	10

Table 1: Cardinalità degli insiemi nell'analisi d'impatto

Metrica	**Formula**	**Valore**
Recall	$\frac{\lvert DIS\rvert}{\lvert AIS\rvert}$	0.40 (40%)
Precision	$\frac{\lvert DIS\rvert}{\lvert CIS\rvert}$	$\approx$ 0.44 (44.4%)

Table 2: Precision e Recall calcolati sull'analisi d'impatto