



Università degli Studi di Salerno

Dipartimento di Informatica

---

Corso di Ingegneria del Software: Tecniche Avanzate

# CodeSmile



**Change Request Document**  
Versione 1.0

## Team

Choaib Goumri  
Mattia Gallucci

---

Anno Accademico 2025–2026

# Indice

<b>1</b>	<b>Introduzione</b>	<b>1</b>
1.1	Contesto e Problematica . . . . .	1
1.2	Architettura del Sistema . . . . .	1
1.2.1	Core Backend (Analisi AST) . . . . .	1
1.2.2	Web Infrastructure (Microservizi) . . . . .	2
<b>2</b>	<b>Change Requests</b>	<b>3</b>
2.1	CR1: Stabilità e Accessibilità dell'Ambiente (Docker) . . . . .	3
2.1.1	Descrizione . . . . .	3
2.1.2	Il Problema . . . . .	3
2.1.3	Soluzione proposta . . . . .	3
2.2	CR2: Feedback Visivo di Avanzamento (Progress Bar) . . . . .	4
2.2.1	Descrizione . . . . .	4
2.2.2	Il Problema . . . . .	4
2.2.3	Soluzione proposta . . . . .	4
2.3	CR3: Rilevamento Code Smell : Deep Nesting . . . . .	4
2.3.1	Descrizione . . . . .	4
2.3.2	Il Problema . . . . .	4
2.3.3	Soluzione proposta . . . . .	4
2.4	CR4: Rilevamento Code Smell : Dead Code . . . . .	5
2.4.1	Descrizione . . . . .	5
2.4.2	Il Problema . . . . .	5
2.4.3	Soluzione proposta . . . . .	5
2.5	CR5: Nuova Metrica di "Densità degli Smell" . . . . .	5
2.5.1	Descrizione . . . . .	5
2.5.2	Il Problema . . . . .	5
2.5.3	Soluzione proposta . . . . .	5
2.6	Riepilogo Stato Modifiche . . . . .	6

# 1 Introduzione

## 1.1 Contesto e Problematica

L'avvento dei *ML-enabled systems* ha introdotto nuove sfide di manutenzione rispetto al software tradizionale. La pressione sul *time-to-market* favorisce l'accumulo di **AI-Specific Technical Debt**, compromettendo stabilità, riproducibilità ed evoluzione. Questo debito si manifesta attraverso i **Machine Learning Specific Code Smells (ML-CSs)**, difetti architetturali o implementativi legati all'uso improprio di librerie (Pandas, PyTorch, TensorFlow).

### Soluzione Proposta

**CodeSmile** è un tool di analisi statica per progetti Python che mira a ridurre il debito tecnico identificando 16 tipologie di smell in quattro aree critiche: *Data Debt*, *Model Debt*, *Configuration Debt* ed *Ethics Debt*.

## 1.2 Architettura del Sistema

Il sistema adotta un design modulare orientato agli oggetti e si divide in due macro-ambiti: il core (Python Package) e l'infrastruttura Web (Microservizi).

### 1.2.1 Core Backend (Analisi AST)

Basato sull'analisi degli *Abstract Syntax Trees* (AST), è strutturato nei seguenti package:

- **cli**: Gestisce l'interfaccia a riga di comando e l'orchestrazione (`cli_runner`).
- **code\_extractor**: Estrae semantica da DataFrames, librerie, modelli e variabili.
- **components**: Cuore operativo. Include l'`Inspector` (analisi file), il `Project Analyzer` (gestione progetto) e il `Rule Checker`.
- **detection\_rules**: Contiene la gerarchia delle regole (`Smell`), divise in *API-Specific* (es. errori Pandas) e *Generic*.
- **gui & report**: Gestione interfaccia desktop (Tkinter) e generazione output (CSV/Excel).

### 1.2.2 Web Infrastructure (Microservizi)

Architettura distribuita con Frontend in React/Next.js e Backend Python basato su FastAPI:

- **API Gateway:** Punto di ingresso unico, gestisce routing e sicurezza.
- **AI Analysis Service:** Rilevamento semantico tramite LLM.
- **Static Analysis Service:** Esegue l'analisi AST riutilizzando la logica core dell'Inspector.
- **Report Service:** Aggrega i dati e genera statistiche visualizzabili.

## 2 Change Requests

Questo capitolo descrive le richieste di modifica (CR) identificate per l'evoluzione del sistema CodeSmile. Le richieste coprono miglioramenti infrastrutturali, ottimizzazioni dell'esperienza utente ed estensioni delle capacità di analisi del software.

### 2.1 CR1: Stabilità e Accessibilità dell'Ambiente (Docker)

**Tipologia di manutenzione:** Correttiva.

#### 2.1.1 Descrizione

L'obiettivo di questa richiesta è garantire che l'ambiente di sviluppo e distribuzione (basato su container) sia stabile, facilmente installabile e accessibile su qualsiasi macchina ospite senza configurazioni manuali complesse.

#### 2.1.2 Il Problema

Attualmente, l'avvio del sistema può presentare problemi di comunicazione tra i vari servizi (Web App e API) e la macchina dell'utente, rendendo talvolta inaccessibile l'interfaccia grafica. Inoltre, i tempi di installazione delle dipendenze possono risultare lunghi o incoerenti tra diversi ambienti di sviluppo.

#### 2.1.3 Soluzione proposta

Si propone di standardizzare la configurazione di rete dei container per permettere una connessione immediata "out-of-the-box". Verranno inoltre ottimizzati i processi di costruzione del software per garantire che ogni sviluppatore o utente utilizzi esattamente le stesse versioni delle librerie, eliminando errori dovuti ad ambienti disallineati.

## **2.2 CR2: Feedback Visivo di Avanzamento (Progress Bar)**

**Tipologia di manutenzione:** Perfettiva.

### **2.2.1 Descrizione**

Questa richiesta mira a introdurre una barra di avanzamento nell'interfaccia grafica (GUI) per comunicare all'utente lo stato dell'analisi in tempo reale, migliorando l'usabilità del sistema.

### **2.2.2 Il Problema**

Durante l'analisi di progetti di grandi dimensioni, l'interfaccia attuale non fornisce un feedback chiaro sulla percentuale di completamento. L'utente vede solo dei log testuali o, in alcuni casi, l'interfaccia può sembrare "bloccata", creando incertezza sul fatto che il programma stia effettivamente lavorando.

### **2.2.3 Soluzione proposta**

Verrà integrato un indicatore visivo (barra di progresso) che mostra chiaramente quanto manca alla fine dell'operazione. Il sistema calcolerà il totale dei file da esaminare e aggiornerà la barra passo dopo passo, migliorando la percezione delle prestazioni e la trasparenza del processo.

## **2.3 CR3: Rilevamento Code Smell : Deep Nesting**

**Tipologia di manutenzione:** Evolutiva.

### **2.3.1 Descrizione**

Si richiede l'aggiunta di una nuova regola di analisi per identificare porzioni di codice eccessivamente complesse e annidate (es. troppe condizioni dentro altre condizioni). Questa modifica estende le capacità di detection del tool.

### **2.3.2 Il Problema**

Il codice che presenta troppi livelli di profondità (ad esempio, molti "if" o cicli uno dentro l'altro) diventa difficile da leggere, testare e mantenere. Attualmente il sistema non segnala specificamente questo difetto strutturale.

### **2.3.3 Soluzione proposta**

Il sistema verrà istruito per rilevare la profondità logica di ogni funzione. Se questa supera una soglia di tolleranza (che indica una cattiva leggibilità), verrà generata una segnalazione specifica ("Deep Nesting Smell"), suggerendo allo sviluppatore di semplificare la struttura del codice.

## 2.4 CR4: Rilevamento Code Smell : Dead Code

**Tipologia di manutenzione:** Evolutiva.

### 2.4.1 Descrizione

Questa modifica introduce la capacità di rilevare automaticamente il “codice morto”, ovvero istruzioni presenti nel progetto ma che non verranno mai eseguite, aggiungendo una nuova funzionalità di pulizia del codice.

### 2.4.2 Il Problema

I progetti software accumulano spesso codice inutile (residui di vecchie funzionalità, variabili non usate, o istruzioni poste dopo un termine di funzione). Questo “rumore” riduce la manutenibilità e può confondere chi legge il codice.

### 2.4.3 Soluzione proposta

Verrà implementata una nuova regola che analizza il flusso logico del programma per individuare parti irraggiungibili o inutili. Il sistema segnalerà queste righe permettendo agli sviluppatori di ripulire il progetto in sicurezza, riducendone la dimensione e migliorandone la chiarezza.

## 2.5 CR5: Nuova Metrica di “Densità degli Smell”

**Tipologia di manutenzione:** Evolutiva.

### 2.5.1 Descrizione

Si propone l'introduzione di una metrica statistica che metta in relazione il numero di errori rilevati con la dimensione del file analizzato, offrendo una nuova prospettiva analitica sui dati raccolti.

### 2.5.2 Il Problema

Attualmente, il sistema riporta il numero assoluto di problemi. Tuttavia, 10 errori in un file di 100 righe sono molto più gravi di 10 errori in un file di 10.000 righe. Senza questo contesto, è difficile per i manager e gli sviluppatori capire quali file richiedono un intervento prioritario.

### 2.5.3 Soluzione proposta

Il sistema calcolerà un indice di “Densità” (Numero di problemi / Dimensioni del codice). Questo valore permetterà di classificare i file e i progetti in base alla loro qualità reale, fornendo una visione immediata delle aree più critiche (es. densità Bassa, Media, Alta) nei report finali.

## 2.6 Riepilogo Stato Modifiche

ID	Titolo	Tipo Manutenzione	Priorità	Stato
CR1	Stabilità Ambiente Docker	Correttiva	Alta	In Analisi
CR2	Progress Bar GUI	Perfettiva	Media	In Analisi
CR3	Smell: Deep Nesting	Evolutiva	Alta	In Analisi
CR4	Smell: Dead Code	Evolutiva	Media	In Analisi
CR5	Metrica: Densità Smell	Evolutiva	Alta	In Analisi

*Tabella 2.1: Tabella riassuntiva delle Change Requests*