



Università degli Studi di Salerno

Corso di Ingegneria del Software: Tecniche Avanzate

CodeSmile

Report Finale

Versione 1.0

Team:

Antonio Caiazzo
Emanuele Iovane
Salvatore Di Martino



Progetto: CodeSmile	Versione: 1.0
Documento: Report Finale	Data: 09/01/2026

Partecipanti:

Nome	Matricola
Antonio Caiazzo	NF22500205
Salvatore Di Martino	NF22500114
Emanuele Iovane	NF22500162

Scritto da:	Antonio Caiazzo, Salvatore Di Martino, Emanuele Iovane
--------------------	--

Progetto: CodeSmile	Versione: 1.0
Documento: Report Finale	Data: 09/01/2026

Indice

1. Executive Summary	4
1.1. Risultati Chiave	4
2. Contesto e Baseline del Sistema	4
2.1. Architettura CLI/GUI	4
2.2. Architettura WebApp (gateway-services)	4
2.3. Modalità di utilizzo	4
3. Change Requests Implementate	5
4. Analisi di impatto	6
4.1. Metodologia	6
4.2. Sintesi dell'Analisi di Impatto	6
5. Strategia di Testing e Risultati	6
5.1. Master Test Plan	6
5.2. Pre-modification Testing	7
5.3. Post-modification Testing	7
5.4. Regression testing	7
6. Conclusioni e Sviluppi Futuri	7

Progetto: CodeSmile	Versione: 1.0
Documento: Report Finale	Data: 09/01/2026

1. Executive Summary

Questo documento consolida l'intero ciclo di vita del progetto CodeSmile, dalla descrizione iniziale dell'architettura fino alla consegna a valle delle Change Requests (CR01-CR03). L'obiettivo è fornire una vista unitaria e tracciabile dell'evoluzione del sistema, delle analisi effettuate e delle evidenze di qualità (impact analysis e testing).

1.1. Risultati Chiave

- **Portabilità migliorata** della WebApp tra esecuzione locale e container (CR01), riducendo interventi manuali su import ed endpoint.
- **Arricchimento della pipeline CLI** con generazione automatica del Call Graph e produzione degli artefatti call_graph.json e call_graph.dot (CR02).
- **Introduzione nella WebApp di una visualizzazione interattiva del Call Graph** con integrazione degli smell rilevati via analisi statica (CR03).
- **Validazione post-modifica e regressione superata:** 215 test complessivi eseguiti, 215 superati, con esclusioni previste per moduli AI e GUI.

2. Contesto e Baseline del Sistema

CodeSmile è uno strumento di static code analysis orientato all'individuazione di ML-specific code smells in progetti Python. Il sistema combina regole basate su AST con componenti di orchestrazione (CLI/GUI) e una WebApp a microservizi (gateway, AI Analysis, Static Analysis, Report). La baseline del progetto è composto da due macro-ambienti: (i) l'esecuzione locale via CLI/GUI, e (ii) la fruizione via WebApp. L'architettura a package autonomi riduce l'accoppiamento e abilita interventi evolutivi mirati, mantenendo tracciabilità tra requisiti, componenti e test.

2.1. Architettura CLI/GUI

Il core locale comprende la pipeline di analisi orchestrata dal runner, i componenti di analisi (Inspector e RuleChecker), gli extractor semanticci e il modulo di reportistica. I package principali includono: cli, components, detection_rules, code_extractor e report.

2.2. Architettura WebApp (gateway-services)

La WebApp segue un'architettura gateway-services basata su FastAPI. Il gateway espone gli endpoint e inoltra le richieste ai servizi: AI Analysis Service, Static Analysis Service e Report Service. Ogni servizio mantiene una struttura a livelli (main, routers, utils) per separare avvio, API e logica applicativa.

2.3. Modalità di utilizzo

- **CLI:** analisi di singoli file o progetti, con opzioni di parallelismo (walkers), gestione input/output e ripresa (resume).
- **GUI:** configurazione guidata e visualizzazione log in tempo reale.
- **WebApp:** upload di file o progetti e consultazione dei risultati (lista smell e report).

Progetto: CodeSmile	Versione: 1.0
Documento: Report Finale	Data: 09/01/2026

3. Change Requests Implementate

Le Change Requests sono state definite per migliorare qualità architetturale, capacità analitiche e user experience. La tabella seguente fornisce una sintesi; le sottosezioni descrivono obiettivi e risultato ottenuto.

ID	Titolo	Categoria	Esito
CR01	Allineamento dinamico import ed endpoint tra locale e Docker	Preventive	Rilasciata
CR02	Generazione automatica del Call Graph nella CLI	Perfective	Rilasciata
CR03	Call graph interattivo in WebApp con visualizzazione smell (analisi statica)	Perfective	Rilasciata

CR01 - Allineamento Locale vs Container

Obiettivo: eliminare la necessità di modifiche manuali alla codebase per passare tra esecuzione locale e containerizzata. Intervento: standardizzazione degli import, parametrizzazione degli endpoint tramite variabili d'ambiente, aggiornamento del build context e replica della struttura dei package nei container.

CR02 - Generazione Call Graph in CLI

Obiettivo: estendere la CLI con un comando/flag che abiliti la generazione automatica del call graph durante l'analisi, producendo un artefatto strutturale (call_graph.json e call_graph.dot) integrabile nel processo di reportistica. Intervento principale: introduzione/isolamento della logica di generazione del call graph in un componente dedicato (CallGraphGenerator).

CR03 – Call Graph Interattivo in WebApp

Obiettivo: trasformare la visualizzazione dei risultati da lista statica a rappresentazione grafica e semantica, consentendo navigazione tra nodi e visualizzazione del codice sorgente associato. Intervento: nuova pagina front-end per il grafo, estensioni backend per produrre dati utili all'interazione, e gestione upload efficiente di progetti (ZIP).

Progetto: CodeSmile	Versione: 1.0
Documento: Report Finale	Data: 09/01/2026

4. Analisi di impatto

Per ciascuna CR è stata condotta un Impact Analysis basata su analisi statica della codebase. Il processo utilizza call graph e matrice di raggiungibilità per ricostruire dipendenze e propagazione, derivando in modo tracciabile gli insiemi SIS/CIS e confrontandoli con le modifiche effettive (AIS).

4.1. Metodologia

- Uso dei **Call graph**: evidenzia il flusso di invocazioni e l'accoppiamento tra componenti, a un livello di astrazione coerente con la CR.
- Uso delle **Matrice di raggiungibilità**: quantifica dipendenze dirette (1) e indirette (2+).
- **Insiemi di impatto**: SIS (starting), CIS (candidate), AIS (actual), FPIS (false positive), DIS (discovered).
- Valutazione attraverso l'uso delle **Metriche**: **precision** e **recall** misurano la qualità della previsione dell'impatto.

4.2. Sintesi dell'Analisi di Impatto

CR	SIS	CIS	AIS	Precision	Recall
CR01	14 file	14 file	14 file	1.00	1.00
CR02	3 file	3 file	3 file	0.67	0.67
CR03	8 file (SIS) + 1 (CIS)	9 file	11 file	0.88	0.72

- **CR01**: impatto confinato a gateway, servizi e configurazione Docker; nessun falso positivo e nessun componente emerso in corso d'opera.
- **CR02**: previsione prudente ma con un falso positivo (Inspector) e la necessità di introdurre un nuovo modulo dedicato (CallGraphGenerator).
- **CR03**: previsione accurata sui componenti centrali, ma con impatti emersi su componenti condivisi e di front-end (Navbar e gestione ZIP).

5. Strategia di Testing e Risultati

Il processo di verifica è stato strutturato su quattro livelli: test di unità, integrazione, sistema e end-to-end (WebApp). La baseline di regressione è stata definita tramite system testing pre-modifica (Category Partition). A valle delle CR, sono stati eseguiti post-modification testing e regression testing sull'intera codebase, con esclusioni pianificate per moduli AI e GUI.

5.1. Master Test Plan

Per il testing di unità si è usato un approccio white-box con attenzione alla branch coverage.

Si è deciso di procedere con il test di integrazione per la validazione delle interazioni tra UI (CLI/GUI), Inspector, RuleChecker e Report_Generator. Per il testing di sistema si è utilizzato un approccio black-box con l'uso di Category Partition Method, per esplorare combinazioni di input/configurazioni. Inoltre, per la web app, si sono automatizzati i test con Cypress, eseguendo testing E2E, per i diversi flussi utente.

Progetto: CodeSmile	Versione: 1.0
Documento: Report Finale	Data: 09/01/2026

5.2. Pre-modification Testing

Prima dell'introduzione delle CR è stata eseguita una suite di system test derivata con Category Partition Method. I parametri coprono: numero/estensione dei file, struttura del progetto, presenza e tipologia degli smell, modalità di esecuzione (CLI/WebApp), parallelismo e walkers, resume, output path, e disponibilità backend per la WebApp. Le combinazioni sono state selezionate imponendo una sola sorgente di errore per test frame, per semplificare diagnosi.

5.3. Post-modification Testing

Le verifiche post-modifica hanno validato funzionalità introdotte e componenti modificate dalle tre CR. Per CR02 e CR03 sono stati aggiunti test specifici su più livelli (unità, integrazione, sistema, frontend ed E2E). Per CR02 è stata introdotta la categoria di configurazione CG (Call Graph) nel category partition e sono stati aggiunti i test case TC_21-TC_28.

5.4. Regression testing

La regressione ha previsto la riesecuzione completa delle suite backend (unit e integration) e frontend (Jest), oltre ai test E2E Cypress e alla baseline di system test (TC_01-TC_20), estesa con i nuovi casi fino a TC_28.

Categoria	Test totali	Superati	Falliti	Successo
Backend Unit	197	197	0	100%
Backend Integration	5	5	0	100%
Gateway Integration	5	5	0	100%
Frontend Unit (Jest)	29	29	0	100%
Frontend E2E (Cypress)	23	23	0	100%
System Tests	28	28	0	100%

Totale: 287 test eseguiti, 287 superati (stato complessivo: SUPERATO, con esclusioni previste per moduli AI e GUI).

6. Conclusioni e Sviluppi Futuri

Il progetto ha consolidato un processo di evoluzione controllata basato su change management, impact analysis e quality assurance. Le tre CR hanno aumentato portabilità, osservabilità strutturale e valore per l'utente finale, mantenendo la stabilità del baseline. Per gli sviluppi futuri individuali, si sono individuati diversi punti:

- Integrazione più stretta tra call graph e reportistica: metriche di centralità, percorsi critici e suggerimenti di refactoring.
- Estensione della visualizzazione WebApp con filtri e drill-down su gli smell (gravità, categoria, file).
- Automazione completa del quality gate (coverage e test) su pipeline CI.
- Estensione del catalogo di smell e supporto a nuovi framework/librerie ML.
- Ottimizzazione performance per progetti di grandi dimensioni (caching e incremental analysis).