



UNIVERSITÀ DEGLI STUDI  
DI SALERNO

# CodeSmile

**Machine Learning-Specific Code Smell Detection Tool**

Corso di Ingegneria, Gestione ed Evoluzione del Software  
A.A. 2024/2025

Docente

**Prof Andrea De Lucia**

Tutor progetto

**Dott. Gilberto Recupito**

*CodeSmile*

# Chi siamo



**Simone Silvestri**

Software Engineering & IT  
Management

**0522501419**



**Matteo Ercolino**

Software Engineering & IT  
Management

**0522501462**

# Indice

1. Introduzione
2. Descrizione sistema esistente
3. Change Requests
4. Testing
5. Sistema evoluto



# Introduzione

**CodeSmile** è una suite di strumenti per identificare e gestire code smell specifici nel codice Python di Machine Learning. Permette di analizzare un intero progetto o un singolo file effettuando:

- **Analisi Statica:** Rilevamento basato su regole e analisi AST.
- **Analisi Basata su AI:** Sperimentazione con LLM per l'individuazione di smell.

In entrambe le tipologie di analisi permette di individuare 16 tipi diversi di code smell.

Generic Code Smells

Name	Description
Broadcasting Feature Not Used	Tensor operations that fail to utilize TensorFlow's broadcasting feature.
Columns and DataType Not Explicitly Set	DataFrames created without explicitly setting column names and data types.
Deterministic Algorithm Option Not Used	This smell occurs when the option torch.use_deterministic_algorithms(True) is not removed.
Empty Column Misinitialization	Initializing DataFrame columns with zeros or empty strings.
Hyperparameters Not Explicitly Set	Missing explicit hyperparameter definitions for ML models.
In-Place APIs Misused	Assuming Pandas methods modify DataFrames in-place without reassignment.
Memory Not Freed	Failing to free memory for ML models declared in loops.
Merge API Parameter Not Explicitly Set	Missing explicit how and on parameters in Pandas merge operations.
NaN Equivalence Comparison Misused	Incorrect comparison of values with np.nan.
Unnecessary Iteration	Using explicit loops instead of Pandas vectorized operations.

# Introduzione

**CodeSmile** è una suite di strumenti per identificare e gestire code smell specifici nel codice Python di Machine Learning. Permette di analizzare un intero progetto o un singolo file effettuando:

- **Analisi Statica:** Rilevamento basato su regole e analisi AST.
- **Analisi Basata su AI:** Sperimentazione con LLM per l'individuazione di smell.

In entrambe le tipologie di analisi permette di individuare 16 tipi diversi di code smell.

API Specific Code Smells

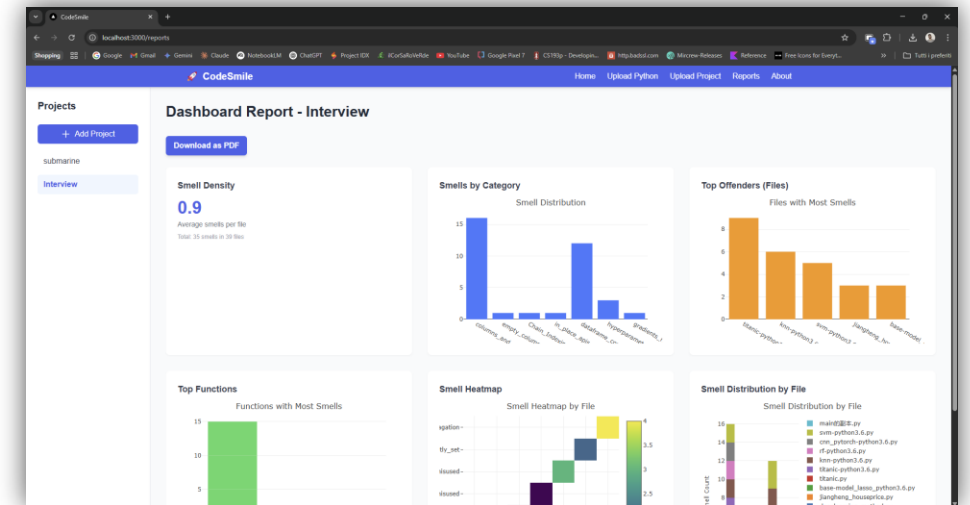
Name	Description
Chain Indexing	Inefficient use of chained indexing in Pandas DataFrames (df["col"][0]).
DataFrame Conversion API Misused	Using .values() to convert Pandas DataFrames instead of .to_numpy().
Gradients Not Cleared	Missing optimizer.zero_grad() before backward propagation in PyTorch.
Matrix Multiplication API Misused	Misusing NumPy's np.dot() for matrix multiplication.
PyTorch Call Method Misused	Direct use of self.net.forward() instead of calling self.net() in PyTorch.
TensorArray Not Used	Using tf.constant() inefficiently in loops instead of tf.TensorArray().

# Descrizione sistema esistente

Il sistema integra tre modalità di utilizzo:

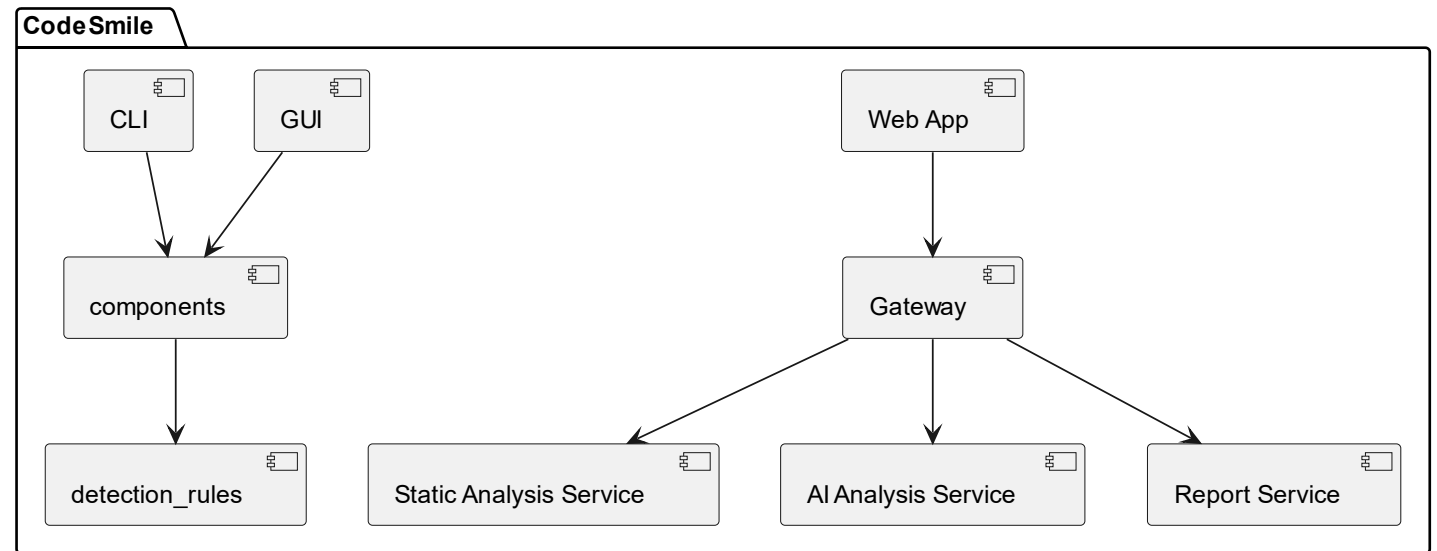
- **CLI:** per analisi rapide e automatizzabili.
- **GUI desktop:** realizzata con Tkinter, pensata per un'esecuzione semplice senza terminale.
- **Web App interattiva:** basata su architettura gateway-services, che consente l'upload di progetti e la visualizzazione dei risultati online.

Alla base di tutte le modalità vi è il core di analisi che sfrutta AST e modelli AI, con le 16 regole di rilevamento.



# Architettura

- CodeSmile è costruito su **un'architettura modulare** a pacchetti indipendenti, progettata per favorire **scalabilità** e **manutenibilità**.
- Il **core engine** gestisce l'analisi si interfaccia sia con strumenti locali (CLI e GUI) sia con la Web App.
- Questa architettura garantisce già buona separazione delle responsabilità e basso accoppiamento fra i moduli, ponendo solide basi per le **successive evoluzioni**.



# Flusso di utilizzo

- L'utente ha a disposizione **tre modalità di accesso**:
  - **CLI** per avviare analisi tramite terminale, con opzioni configurabili.
  - **GUI** per eseguire analisi locali tramite un'interfaccia desktop intuitiva.
  - **Web App** per caricare file o progetti direttamente da **browser**.
- In tutti i casi, il flusso è il medesimo:
  - **Selezione** dei file o del progetto da analizzare.
  - Scelta della **modalità di analisi**: Statica (AST-based) oppure AI-Based (sperimentale).
  - **Esecuzione** del rilevamento e visualizzazione ed **esportazione** dei risultati (CSV).

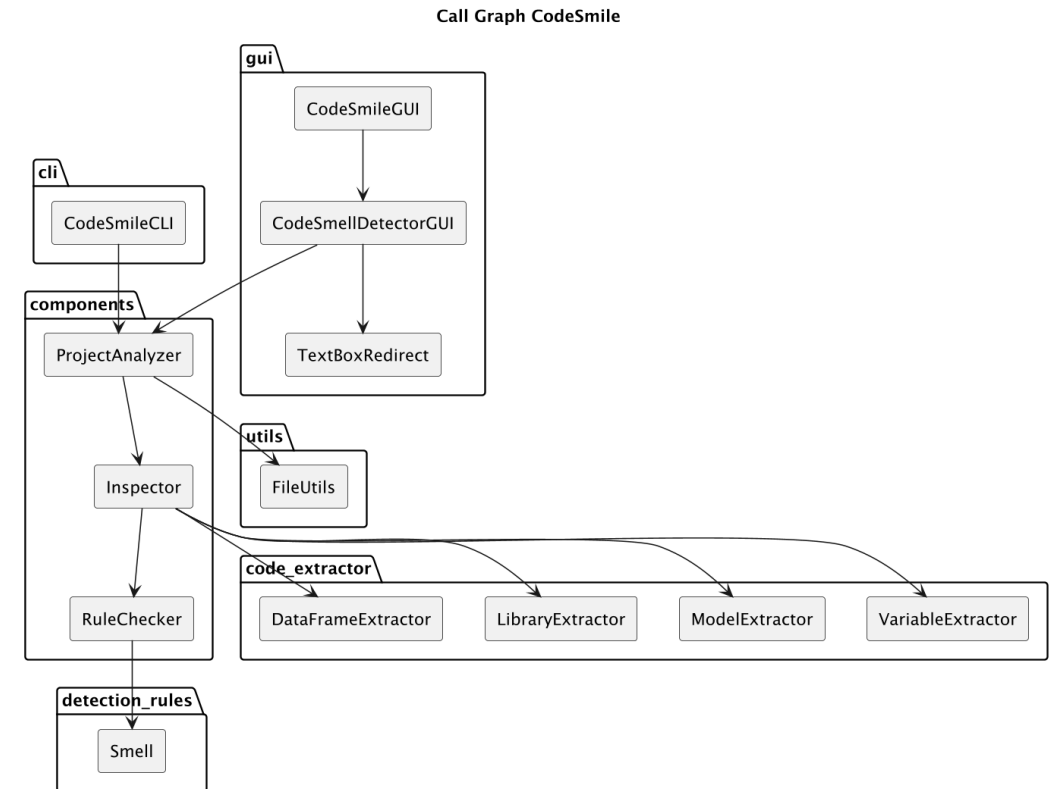




# Reverse engineering

Per supportare l'evoluzione del sistema, è stata condotta un'attività di reverse engineering sul codice esistente:

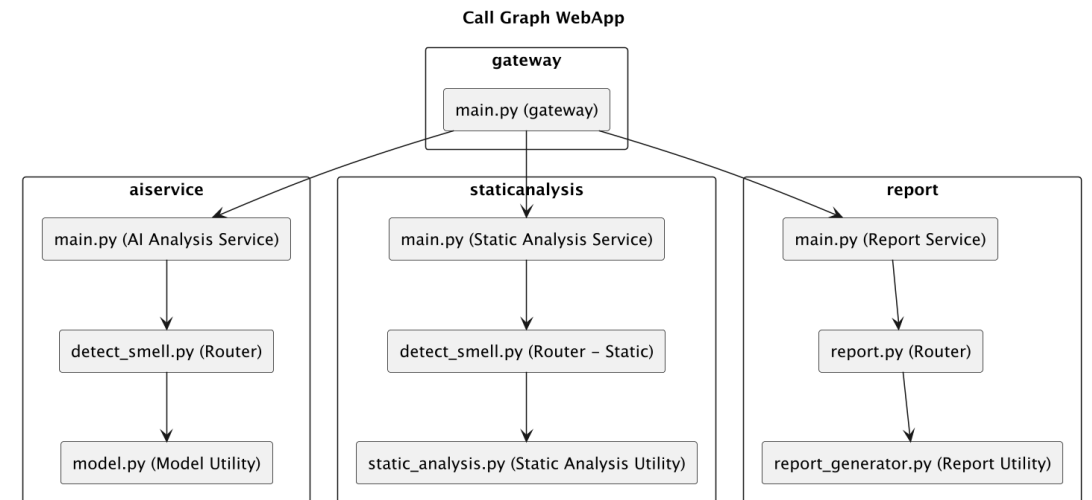
- **Call Graph del core tool** (CodeSmile CLI/GUI): per mappare le relazioni tra moduli e classi principali.
- **Call Graph della Web App**: per analizzare l'interazione tra Gateway e Microservizi.



# Reverse engineering

Per supportare l'evoluzione del sistema, è stata condotta un'attività di reverse engineering sul codice esistente:

- **Call Graph del core tool** (CodeSmile CLI/GUI): per mappare le relazioni tra moduli e classi principali.
- **Call Graph della Web App**: per analizzare l'interazione tra Gateway e Microservizi.



# Reverse engineering

Inoltre, è stata costruita una **Reachability Matrix** per misurare:

- Le dipendenze **dirette** (invocazioni immediate).
- Le dipendenze **indirette** (propagazione su più livelli).

	CodeSmileCLI	CodeSmellDetectorGUI	CodeSmileGUI	TextBoxRedirect	ProjectAnalyzer	Inspector	RuleChecker	DataFrameExtractor	LibraryExtractor	ModelExtractor	VariableExtractor	Smell	FileUtils
CodeSmileCLI	0	0	0	0	1	2	3	3	3	3	3	4	2
CodeSmellDetectorGUI	0	0	0	1	1	2	3	3	3	3	3	4	2
CodeSmileGUI	0	1	0	2	2	3	4	4	4	4	4	5	3
TextBoxRedirect	0	0	0	0	0	0	0	0	0	0	0	0	0
ProjectAnalyzer	0	0	0	0	0	1	2	2	2	2	2	3	1
Inspector	0	0	0	0	0	0	1	1	1	1	1	2	0
RuleChecker	0	0	0	0	0	0	0	0	0	0	0	1	0
DataFrameExtractor	0	0	0	0	0	0	0	0	0	0	0	0	0
LibraryExtractor	0	0	0	0	0	0	0	0	0	0	0	0	0
ModelExtractor	0	0	0	0	0	0	0	0	0	0	0	0	0
VariableExtractor	0	0	0	0	0	0	0	0	0	0	0	0	0
Smell	0	0	0	0	0	0	0	0	0	0	0	0	0
FileUtils	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabella 1.1: Reachability Matrix – CodeSmile

# Reverse engineering

Inoltre, è stata costruita una **Reachability Matrix** per misurare:

- Le dipendenze **dirette** (invocazioni immediate).
- Le dipendenze **indirette** (propagazione su più livelli).

	gateway_main	ai_main	ai_router	ai_utility	static_main	static_router	static_utility	report_main	report_router	report_utility
gateway_main	0	1	2	3	1	2	3	1	2	3
ai_main	0	0	1	2	0	0	0	0	0	0
ai_router	0	0	0	1	0	0	0	0	0	0
ai_utility	0	0	0	0	0	0	0	0	0	0
static_main	0	0	0	0	0	1	2	0	0	0
static_router	0	0	0	0	0	0	1	0	0	0
static_utility	0	0	0	0	0	0	0	0	0	0
report_main	0	0	0	0	0	0	0	0	1	2
report_router	0	0	0	0	0	0	0	0	0	1
report_utility	0	0	0	0	0	0	0	0	0	0

Tabella 1.2: Reachability Matrix - Web App

# Reverse engineering

Inoltre, è stata costruita una **Reachability Matrix** per misurare:

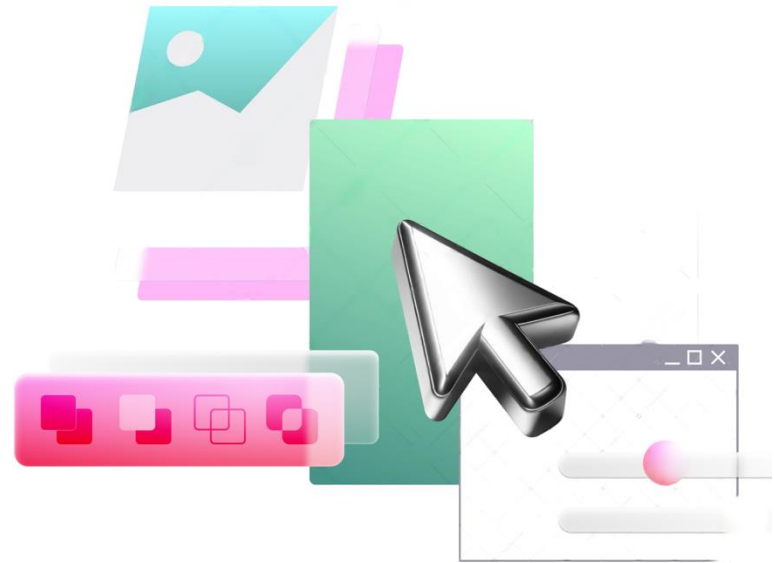
- Le dipendenze **dirette** (invocazioni immediate).
- Le dipendenze **indirette** (propagazione su più livelli).

Questo lavoro ha permesso di:

- Identificare i **punti critici** del sistema (come ProjectAnalyzer e Inspector).
- Stimare **l'impatto delle modifiche** prima della loro implementazione.

# Change Requests

- **CR1 – Dashboard interattiva nella Web App**
- **CR3 – Aggiunta di una Modalità “Quick Scan”**
- **CR2 – Integrazione con strumenti di CI/CD**



• **CR1 – Dashboard interattiva nella Web App** • CR3 – Aggiunta di una Modalità “Quick Scan” • CR2 – Integrazione con strumenti di CI/CD

Aggiungere una dashboard interattiva nella Web App in modo che gli utenti possano visualizzare:

- Metriche avanzate sui code smells rilevati.
- Grafici avanzati

**Obiettivo:** migliorare la visibilità e la comprensione dei problemi.

## **Impact Analysis**

### **Start Impact Set (SIS)**

- app/reports/page.tsx
- components/ChartSection.tsx
- types/types.ts
- gateway\_main.py
- report\_main.py
- report\_router.py
- report\_utility.py



## **Impact Analysis**

**Start Impact Set** (SIS) => 7

**Candidate Impact Set** (CIS) = SIS

- Non è previsto impatto indiretto su altre componenti

## Impact Analysis

**Start Impact Set** (SIS) => 7

**Candidate Impact Set** (CIS) = SIS

### **Actual Impact Set** (AIS)

- app/reports/page.tsx
- types/types.ts,
- components/ChartSection.tsx,
- components/ChartContainer.tsx,
- components/PDFDownloadButton.tsx,
- components/ProjectSidebar.tsx
- components/SmellDensityCard.tsx
- hooks/useReportData.ts
- components/Modal.tsx
- utils/dataFormatters.ts
- report\_main.py
- report\_router.py
- report\_utility.py

## Impact Analysis

**Start Impact Set** (SIS) => 7

**Candidate Impact Set** (CIS) = SIS

**Actual Impact Set** (AIS) => 13

### **Discovered Impact Set** (DIS)

- components/ChartContainer.tsx,
- components/PDFDownloadButton.tsx
- components/ProjectSidebar.tsx
- components/SmellDensityCard.tsx
- hooks/useReportData.ts
- components/Modal.tsx
- utils/dataFormatters.ts

## Impact Analysis

**Start Impact Set** (SIS) => 7

**Candidate Impact Set** (CIS) = SIS

**Actual Impact Set** (AIS) => 13

**Discovered Impact Set** (DIS) => 7

### **False Positive Impact Set (FPIS)**

- gateway\_main.py

• CR1 – Dashboard interattiva nella Web App

• CR3 – Aggiunta di una Modalità “Quick Scan”

• CR2 – Integrazione con strumenti di CI/CD

**Start Impact Set** (SIS) => 7

**Candidate Impact Set** (CIS) = SIS

**Actual Impact Set** (AIS) => 13

**Discovered Impact Set** (DIS) => 7

**False Positive Impact Set** (FPIS) => 1

Precision

$$\frac{|CIS \cap AIS|}{|CIS|} = \frac{6}{7} = 0.85$$

Recall

$$\frac{|CIS \cap AIS|}{|AIS|} = \frac{6}{13} = 0.5$$

Insieme	Contenuto
SIS	app/reports/page.tsx, components/ChartSection.tsx, types/types.ts, gateway_main.py, report_main.py, report_router.py, report_utility.py
CIS	Uguale a SIS
AIS	app/reports/page.tsx, types/types.ts, components/ChartSection.tsx, components/ChartContainer.tsx, components/PDFDownloadButton.tsx, components/ProjectSidebar.tsx, components/SmellDensityCard.tsx, hooks/useReportData.ts, components/Modal.tsx, utils/dataFormatters.ts, report_main.py, report_router.py, report_utility.py
FPIS	gateway_main.py
DIS	components/ChartContainer.tsx, components/PDFDownloadButton.tsx, components/ProjectSidebar.tsx, components/SmellDensityCard.tsx, hooks/useReportData.ts, components/Modal.tsx, utils/dataFormatters.ts
DFS	components/ChartSection.tsx
Precision	0.85
Recall	0.5

Tabella 2.1: Riassunto Insiemi e Metriche – CR1 (Dashboard Interattiva)

• **CR3 – Aggiunta di una Modalità “Quick Scan”**

• CR2 – Integrazione con strumenti di CI/CD

Introdurre una modalità di analisi rapida che consenta agli di controllare solamente i file modificati di recente senza dover eseguire l’analisi dell’intero progetto.

**Obiettivo:** ottimizzare il processo di analisi ripetuta degli stessi progetti..

• **CR3 – Aggiunta di una Modalità “Quick Scan”**

## **Impact Analysis**

### **Start Impact Set (SIS)**

- ProjectAnalyzer
- CodeSmileCLI

## Impact Analysis

**Start Impact Set** (SIS)  $\Rightarrow 2$

### **Candidate Impact Set** (CIS)

- ProjectAnalyzer
- CodeSmileCLI
- Project Repository Cloner



## Impact Analysis

**Start Impact Set** (SIS)  $\Rightarrow$  2

**Candidate Impact Set** (CIS)  $\Rightarrow$  3

### **Actual Impact Set** (AIS)

- ProjectAnalyzer
- CodeSmileCLI
- Project Repository Cloner

## Impact Analysis

**Start Impact Set** (SIS)  $\Rightarrow$  2

**Candidate Impact Set** (CIS)  $\Rightarrow$  3

**Actual Impact Set** (AIS)  $\Rightarrow$  3

**Discovered Impact Set** (DIS)

- Vuoto

**False Positive Impact Set** (FPIS)

- Vuoto

• CR3 – Aggiunta di una Modalità “Quick Scan”

**Start Impact Set** (SIS) => 2

**Candidate Impact Set** (CIS) => 3

**Actual Impact Set** (AIS) => 3

**Discovered Impact Set** (DIS) => 0

**False Positive Impact Set** (FPIS) => 0

**Precision**

$$\frac{|CIS \cap AIS|}{|CIS|} = \frac{3}{3} = 1$$

**Recall**

$$\frac{|CIS \cap AIS|}{|AIS|} = \frac{3}{3} = 1$$

• CR2 – Integrazione con strumenti di CI/CD

Insieme	Contenuto
SIS	ProjectAnalyzer, CodeSmileCLI
CIS	ProjectAnalyzer, CodeSmileCLI, ProjectRepositoryCloner
AIS	ProjectAnalyzer, CodeSmileCLI, ProjectRepositoryCloner
FPIS	{}
DIS	{}
Precision	1.0
Recall	1.0

Tabella 4.1: Riassunto Insiemi e Metriche – CR3 (Quick Scan)

- **CR2 – Integrazione con strumenti di CI/CD**

Introdurre una modalità di analisi rapida che consenta agli di controllare solamente i file modificati di recente senza dover eseguire l'analisi dell'intero progetto.

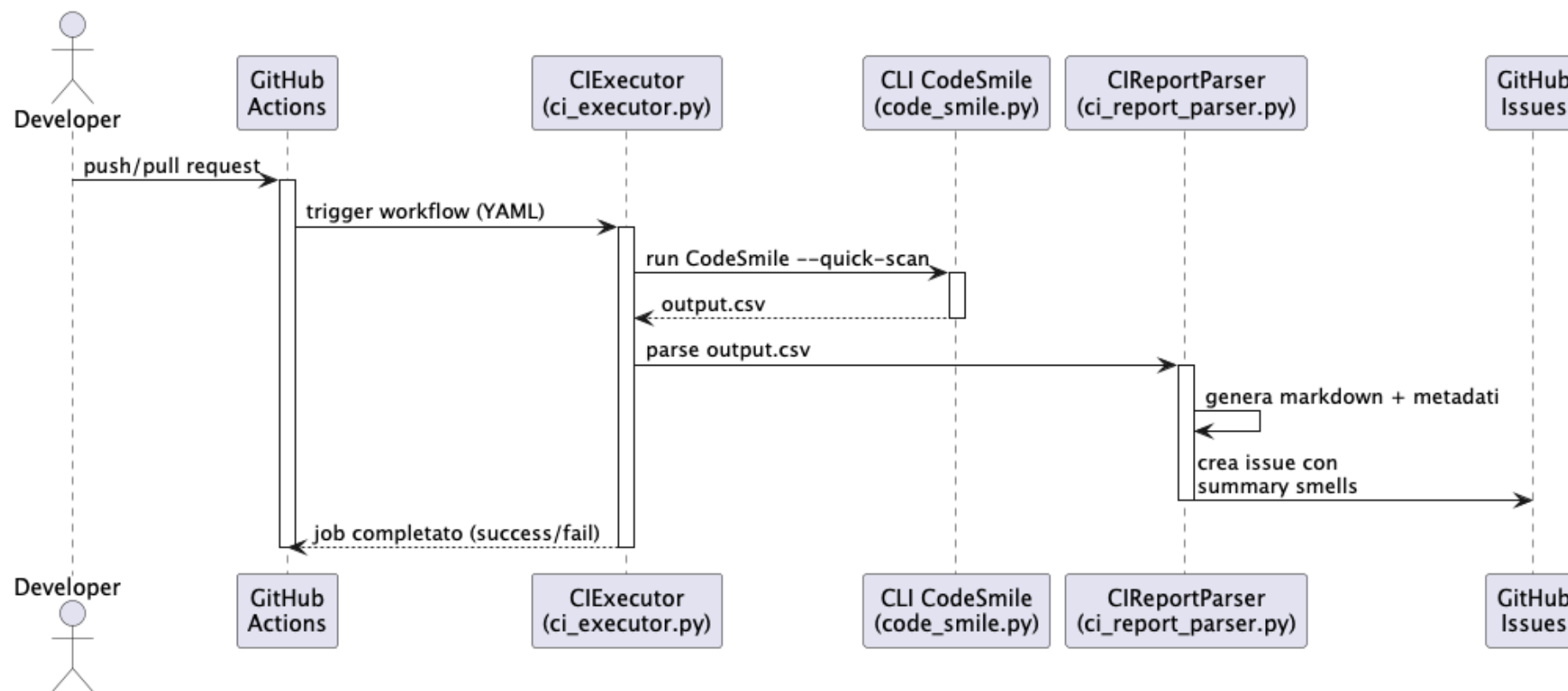
**Obiettivo:** ottimizzare il processo di analisi ripetuta degli stessi progetti.

## Impact Analysis

- Essendo una modifica di tipo **adattiva**, non sono stati previsti impatti sulle classi esistenti. Nessun componente preesistente è stato **modificato o coinvolto**.
- Sono state introdotte due nuove classi per gestire l'integrazione:
  - **CIExecutor**: invoca CodeSmile da linea di comando all'interno della pipeline.
  - **CIReportParser**: analizza l'output e genera automaticamente issue su GitHub.

Non essendoci modifiche a componenti esistenti, gli insiemi di impatto (**SIS, CIS, AIS, FPIS, DIS**) risultano vuoti e le metriche (**Precision, Recall**) non sono definibili. Questa situazione è coerente con il tipo di intervento effettuato.

## • CR2 – Integrazione con strumenti di CI/CD




Sequence diagram – CI support


• CR2 – Integrazione con strumenti di CI/CD

Esempio di **GitHub Issue** creata in automatico con l'esecuzione di CodeSmile:

matthew-2000 opened 3 weeks ago


 Summary

- Analysis date: 11/4/2025
- Total code smells detected: 16

 It is recommended to review the indicated files to improve code quality.

filename	smell_name	line	commit_h
./components/project_analyzer.py	columns_and_datatype_not_explicitly_set	94	<a href="#">fd7a1bd</a>
./components/project_analyzer.py	columns_and_datatype_not_explicitly_set	179	<a href="#">fd7a1bd</a>
./components/project_analyzer.py	columns_and_datatype_not_explicitly_set	270	<a href="#">fd7a1bd</a>
./components/project_analyzer.py	columns_and_datatype_not_explicitly_set	270	<a href="#">fd7a1bd</a>
./test/unit_testing/components/test_project_analyzer.py	columns_and_datatype_not_explicitly_set	69	<a href="#">fd7a1bd</a>
./test/unit_testing/components/test_project_analyzer.py	columns_and_datatype_not_explicitly_set	79	<a href="#">fd7a1bd</a>
./test/unit_testing/components/test_project_analyzer.py	columns_and_datatype_not_explicitly_set	137	<a href="#">fd7a1bd</a>
./test/unit_testing/components/test_project_analyzer.py	columns_and_datatype_not_explicitly_set	208	<a href="#">fd7a1bd</a>
./test/unit_testing/components/test_project_analyzer.py	columns_and_datatype_not_explicitly_set	359	<a href="#">fd7a1bd</a>
./test/unit_testing/components/test_project_analyzer.py	columns_and_datatype_not_explicitly_set	384	<a href="#">fd7a1bd</a>
./test/unit_testing/components/test_project_analyzer.py	columns_and_datatype_not_explicitly_set	401	<a href="#">fd7a1bd</a>
./test/unit_testing/components/test_project_analyzer.py	columns_and_datatype_not_explicitly_set	537	<a href="#">fd7a1bd</a>
./test/unit_testing/components/test_project_analyzer.py	columns_and_datatype_not_explicitly_set	482	<a href="#">fd7a1bd</a>
./test/unit_testing/components/test_project_analyzer.py	columns_and_datatype_not_explicitly_set	474	<a href="#">fd7a1bd</a>
./test/unit_testing/components/test_project_analyzer.py	columns_and_datatype_not_explicitly_set	482	<a href="#">fd7a1bd</a>
./test/unit_testing/components/test_project_analyzer.py	columns_and_datatype_not_explicitly_set	474	<a href="#">fd7a1bd</a>

Create sub-issue



# Considerazioni

Le analisi effettuate hanno mostrato:

- Una buona **precisione** nel prevedere le classi effettivamente modificate, soprattutto in CR3.
- L'introduzione di funzionalità esterne (come in CR2) può **avvenire senza impattare l'architettura esistente.**

Il sistema ha dimostrato una **buona capacità di adattamento ai cambiamenti**, mantenendo coerenza architetturale e manutenibilità.



# Testing

Per assicurare la qualità e la robustezza di CodeSmile, è stato elaborato un **Master Test Plan** che copre **tutti i livelli** di verifica del sistema.

## Unit Testing

Validazione delle singole classi e moduli, con analisi **white-box** e obiettivo **di branch coverage  $\geq 80\%$** .

## Integration Testing

Verifica delle **interazioni** tra componenti principali (es. CLI  $\rightarrow$  Analyzer).

## System Testing

Controllo dell'intero flusso funzionale, dall'input dei file alla generazione dei report, usando il **Category Partition Method**.

## E2E Testing

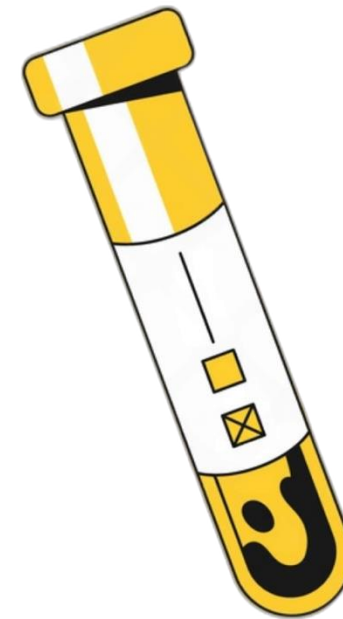
Simulazione dei **principali flussi utente** tramite test Cypress, per assicurare correttezza tra frontend e backend.

# Unit Testing

La suite di test di unità è organizzata in base alla suddivisione in package del sistema. All'interno della directory `test/unit_testing` si trovano sottocartelle che rispecchiano i principali moduli del codice.

**Approccio: white-box testing.** Si analizza la branch coverage di ciascun modulo, verificando che tutti i possibili rami logici del codice vengano effettivamente esercitati durante l'esecuzione dei test.

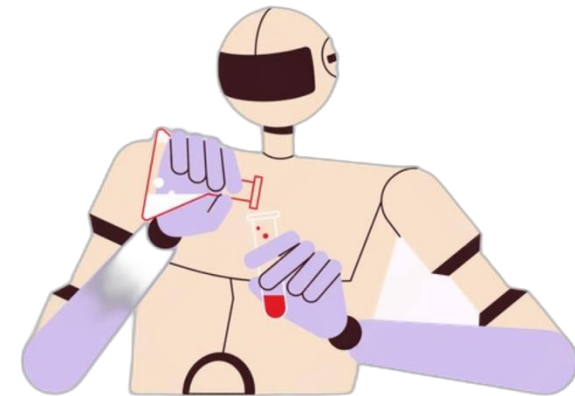
**Obiettivo:** raggiungere (e possibilmente superare) l'80% di branch coverage per ciascun package.



# Integration Testing

L'attività di Integration Testing ha avuto l'obiettivo di validare le interazioni tra i principali componenti architetturali del sistema CodeSmile, con particolare attenzione ai **moduli centrali che compongono l'intero flusso di analisi**: dall'interfaccia utente (sia CLI che GUI), attraverso l'Inspector e il RuleChecker.

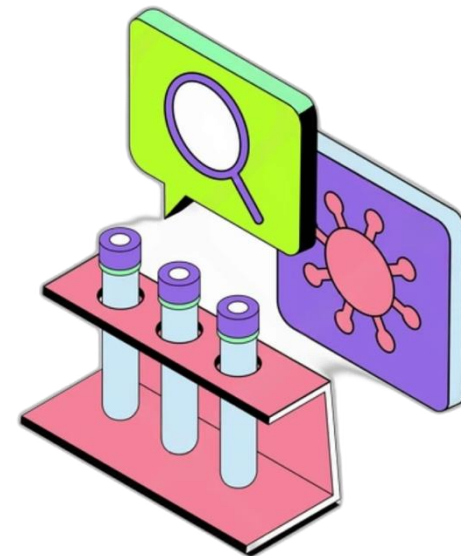
Il testing è stato condotto con approccio **incrementale e top-down**, combinato con tecniche di mocking strategico al fine di isolare i singoli punti di interazione tra i moduli.



# System Testing

**Approccio: testing black-box**, utilizzando casi di test costruiti secondo la tecnica del **Category Partition**, variando parametri come numero e tipo di file, struttura dei progetti e modalità di esecuzione.

- I test sono organizzati in cartelle numerate (TC1, TC2, ecc.) nella directory `system_testing`, contenenti esempi con file anche non .py e directory annidate per simulare scenari reali.
- L'esecuzione è automatizzata tramite lo script `test_system_runner.py`, che simula l'uso da parte di un utente finale.
- Sono stati inoltre inclusi test di robustezza con file non leggibili, interruzioni simulate e cartelle vuote.



# E2E Testing - Webapp

**End-to-End (E2E):** per validare i principali flussi utente, dall'accesso alla homepage fino alla generazione dei report, verificando l'interazione corretta tra front-end, API Gateway e servizi backend. L'automazione è gestita tramite il framework **Cypress**.





I test verificano **la corretta navigazione tra le sezioni della WebApp**, il **caricamento** e **l'analisi** di file Python e progetti, la **robustezza in presenza di errori** (come risposte HTTP 500) con la visualizzazione di messaggi chiari, e la **generazione dei report** con grafici e possibilità di download in PDF.



# Regression testing

**Obiettivo:** garantire che le Change Requests introdotte non abbiano compromesso il comportamento corretto e atteso del sistema esistente. In altre parole, si è mirato a **prevenire regressioni funzionali** e **mantenere inalterata la qualità** del software preesistente.

L'intera suite di test è stata rieseguita con **esito positivo:**

-  166 unit tests
-  8 integration tests
-  21 test cases (system test)
-  20 End-to-End test case (WebApp)

Branch **83%**  
Coverage

# Post-modification testing

**Obiettivo:** perfezionamento e l'estensione dei test di unità e integrazione per tutte le componenti aggiunte o modificate in ciascuna Change Request.

Tutti i test hanno avuto esito positivo, senza rilevazione di anomalie, in conformità con gli **obiettivi definiti** nel Master Test Plan.

Inoltre, ciascuna modifica è stata validata mediante specifici **test funzionali di sistema**.

Branch **87%**  
Coverage



# Conclusioni

Il progetto **CodeSmile** ha dimostrato:

1. un'architettura modulare capace **di assorbire evoluzioni** (CR-01, 02, 03) senza regressioni;
2. un processo di Impact Analysis **affidabile** (precision/recall  $\geq 1.0$  nel caso più critico);
3. una pipeline di testing solida che mantiene il quality gate **sopra l'80 % di coverage**;
4. benefici concreti per gli utenti: **tempi di analisi ridotti** (Quick Scan) e **feedback continuo** (CI/CD).

Il sistema è ora pronto per il possibile rilascio e per **future estensioni** (e.g. nuovi smells o supporto a linguaggi aggiuntionali).





# Grazie per l'attenzione!