Impact Analysis Document

Versione 1.0

CodeSmile

Team Members:
Matteo Ercolino matr. 0522501462
Simone Silvestri matr. 0522501419

Repository: GitHub link

Anno Accademico 2024/2025

Indice

Elenco delle figure

Elenco delle tabelle

Definizioni e Acronimi

SIS (Starting Impact Set): Insieme delle componenti direttamente (o inizialmente) coinvolte dalla Change Request.

CIS (Candidate Impact Set): Insieme delle componenti potenzialmente colpite a livello indiretto.

AIS (Actual Impact Set): Insieme delle componenti realmente modificate durante l'implementazione.

FPIS (False Positive Impact Set): Componenti inizialmente considerate a rischio ma che non subiscono modifiche.

DIS (Discovered Impact Set): Componenti non previste inizialmente ma risultate impattate in corso d'opera.

1 Analisi Statica della Struttura del Sistema

1.1 Obiettivo

Questo capitolo documenta la struttura statica del sistema **CodeSmile**, analizzandone la composizione a livello di classi e servizi, al fine di individuare relazioni di dipendenza che influenzano l'impatto delle modifiche. Le analisi condotte si basano su parsing statico del codice Python, supportato da strumenti di elaborazione automatica dei grafi di chiamata.

1.2 Call Graph - CodeSmile

Il seguente diagramma rappresenta le chiamate tra le principali classi che compongono lo strumento **CodeSmile**, organizzate per package funzionale. Sono riportate unicamente le classi che coordinano o elaborano il processo di analisi dei code smell. Le classi derivate da `Smell` sono state omesse per chiarezza, mantenendo la sola astrazione.

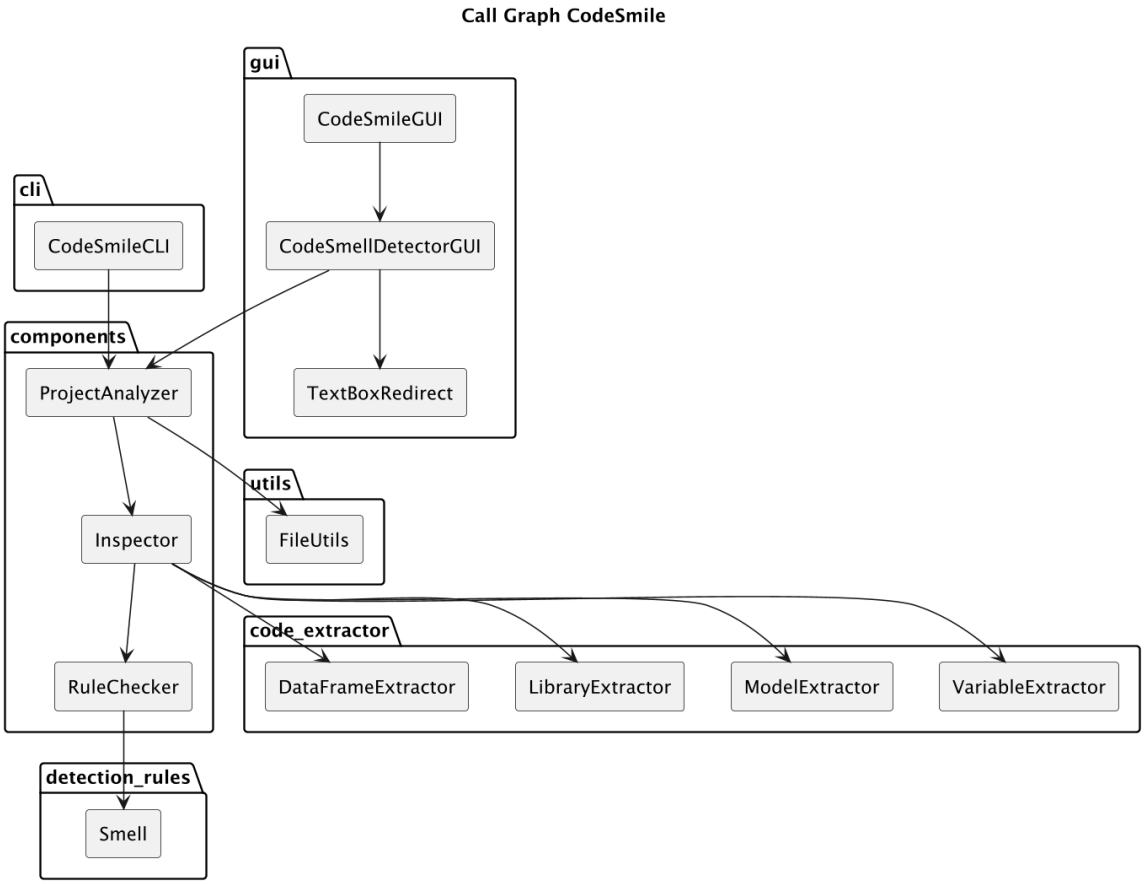

Figura 1.1: Call Graph tra le classi principali di CodeSmile (core tool)

1.3 Call Graph Web App

La figura seguente illustra la struttura e le interazioni tra i microservizi della Web App associata a CodeSmile, organizzata secondo il pattern **gateway-service**. Ogni servizio interno è strutturato

in tre componenti: entrypoint (`main.py`), router e utility.

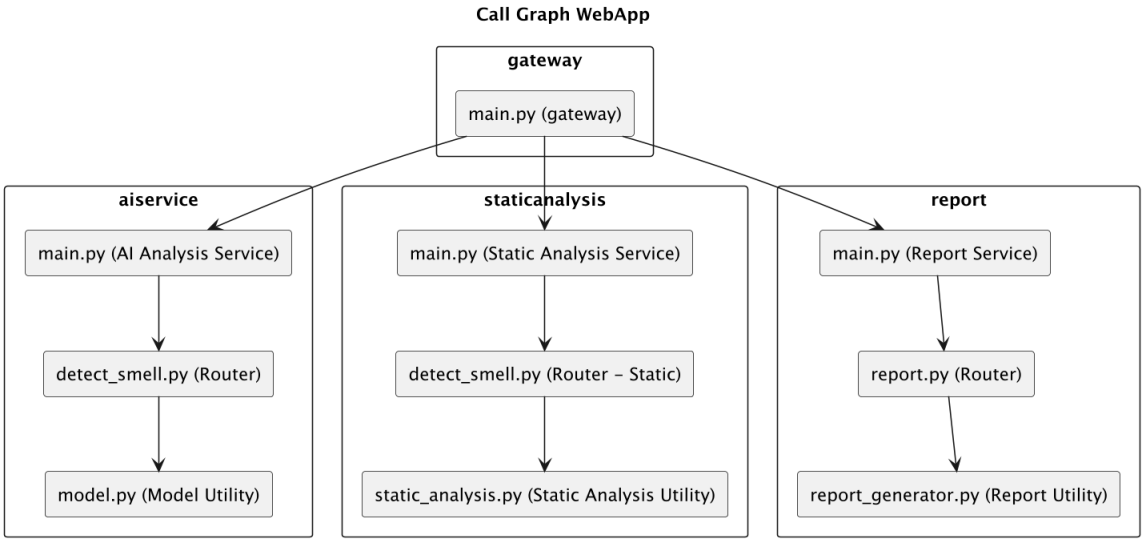

Figura 1.2: Architettura a microservizi e chiamate tra componenti della Web App

Il gateway comunica con ciascun servizio attraverso endpoint REST. I servizi AI e Static Analysis usano moduli separati per orchestrare rispettivamente l'invocazione dei modelli e l'analisi statica. il servizio Report genera un report di dati da poter analizzare.

1.4 Reachability Matrix - CodeSmile

Le seguenti matrici rappresentano la **raggiungibilità** tra le classi principali del sistema. Ogni cella $[i, j]$ indica:

- **0**: nessuna dipendenza tra la classe i e la classe j
- **1**: dipendenza diretta
- **2+**: dipendenza indiretta via uno o più classi intermedie

	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

Legenda dei Colori – Livelli di Dipendenza

1	Dipendenza Diretta
2	Dipendenza Indiretta (1 intermediario)
3	Dipendenza Indiretta (2 intermediari)
4	Dipendenza Indiretta (3 intermediari)
5	Dipendenza Indiretta (4+ intermediari)

Questa matrice evidenzia l'effettiva profondità di propagazione delle chiamate tra le classi centrali del sistema, supportando un'analisi più precisa dell'impatto di eventuali modifiche.

Considerazioni sulla Manutenibilità

La struttura modulare del sistema favorisce la separazione delle responsabilità. Tuttavia, emerge la centralità delle classi `ProjectAnalyzer` e `Inspector`, che coordinano gran parte dell'attività interna del sistema. Ogni modifica che impatta queste classi può potenzialmente propagarsi a tutto il grafo.

1.5 Reachability Matrix – Web App

La seguente matrice rappresenta la raggiungibilità tra i componenti principali della Web App basata su microservizi. Ogni entrypoint dei servizi chiama un router, che a sua volta utilizza una utility.

	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

2 Descrizione CR1 (Dashboard Interattiva nella Web App)

2.1 Analisi di Impatto

La prima Change Request (**CR1**) introduce una **dashboard interattiva** all'interno della Web App, consentendo di visualizzare metriche avanzate sui code smells rilevati in modo interattivo.

2.1.1 Starting Impact Set (SIS)

Le *classi* e i *file* direttamente coinvolti da questa CR, nell'ambito delle macro-componenti menzionate, sono:

- **Frontend (Web App)**:
  – Sarà modificata la pagina dei report già esistente, per includere grafici e metriche interattive, componenti e tipi. In dettaglio:
  - app/reports/page.tsx
  - components/ChartSection.tsx
  - types/types.ts
- **Gateway** (`gateway_main.py`):
  – Potrebbero servire nuovi endpoint o adattamenti di quelli esistenti per fornire i dati alla dashboard.
- **Report Service**:
  – `report_main.py` e `report_router.py` (per gestire la richiesta di metriche aggregate).
  – `report_utility.py` se si dovranno aggiungere metodi utili alla formattazione/aggregazione dei dati da visualizzare.

Pertanto, a livello iniziale, l'insieme delle classi/file potenzialmente toccate dalla CR è:

$$SIS = \{\texttt{app/reports/page.tsx}, \texttt{components/ChartSection.tsx}, \texttt{types/types.ts}, \texttt{gateway\_main.py}, \texttt{report\_main.py}, \texttt{report\_router.py}, \texttt{report\_utility.py}\}$$

2.1.2 Candidate Impact Set (CIS)

In questo caso non sono previsti impatti indiretti su altri componenti, quindi:

$$CIS = SIS$$

2.2 Implementazione delle Modifiche

2.2.1 Actual Impact Set (AIS)

L'implementazione della CR è stata completata. I file effettivamente modificati sono:

$AIS = \{$ `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` $\}$

2.2.2 False Positive Impact Set

Il gateway era stato inizialmente considerato impattato, ma non è stato modificato:

$$FPIS = \{\texttt{gateway\_main.py}\}$$

2.2.3 Discovered Impact Set

Durante l'implementazione sono emersi nuovi file non previsti, come risultato di un refactoring di app/reports/page.tsx per semplificare l'introduzione della nuova dashboard :

$DIS = \{$ `components/ChartContainer.tsx`, `components/PDFDownloadButton.tsx`,
`components/ProjectSidebar.tsx`,
`components/SmellDensityCard.tsx`, `hooks/useReportData.ts`,
`components/Modal.tsx`, `utils/dataFormatters.ts` $\}$

Nota sul Refactoring

Durante l'implementazione della CR, il file `page.tsx` è stato rifattorizzato per migliorarne la modularità, portando alla creazione di nuovi componenti e utility. Sebbene questi file non fossero esplicitamente previsti nella CR, sono stati introdotti esclusivamente per supportare il refactoring di `page.tsx`. Pertanto, sono inclusi nell'**Actual Impact Set (AIS)** e nel **Discovered Impact Set (DIS)**, ma non nel **Starting Impact Set (SIS)**.

2.2.4 Deleted File Set (DFS)

Il seguente file previsto è stato rimosso:

$$DFS = \{\texttt{components/ChartSection.tsx}\}$$

2.3 Metriche

Per valutare la qualità dell'analisi d'impatto, consideriamo le metriche::

$$\text{Precision} = \frac{|CIS \cap AIS|}{|CIS|} = \frac{6}{7} = 0.85 \qquad \text{Recall} = \frac{|CIS \cap AIS|}{|AIS|} = \frac{6}{13} = 0.46$$

L'analisi si è dimostrata abbastanza precisa, ma ha incluso la creazione di nuovi file e un falso positivo: il file `gateway_main.py`, non effettivamente coinvolto.

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	0.46

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

3 Descrizione CR2 (Integrazione con strumenti di CI/CD)

3.1 Analisi di Impatto

La seconda Change Request (**CR2**) ha previsto l'integrazione di CodeSmile con GitHub Actions (o altri strumenti CI/CD), per eseguire controlli automatici sui code smells a ogni push o pull request.

Essendo una modifica di tipo **adattiva**, non sono stati previsti impatti sulle classi esistenti. Nessun componente preesistente è stato modificato o coinvolto.

3.2 Implementazione delle Modifiche

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.

4 Descrizione CR3 (Modalità "Quick Scan")

4.1 Analisi di Impatto

La terza Change Request (**CR3**) introduce una modalità di analisi rapida (*Quick Scan*), limitata ai file effettivamente modificati di recente.

4.1.1 Starting Impact Set (SIS)

A livello di *classi* (in base al Call Graph e alla struttura del progetto), le entità direttamente coinvolte per implementare il *Quick Scan* sono:

- **ProjectAnalyzer** (package `components`):
  – Responsabile dell'analisi del progetto. Dovrà essere esteso per supportare un'analisi parziale su un elenco specifico di file recenti (ottenuto via Git o parametri esterni).
- **Interfaccia (CLI):**
  – **CodeSmileCLI** (`cli`), che dovrà gestire l'opzione `--quick-scan` da linea di comando.

Insieme iniziale dunque:

$$SIS = \{\texttt{ProjectAnalyzer}, \texttt{CodeSmileCLI}\}$$

4.1.2 Candidate Impact Set (CIS)

Potrebbero essere coinvolti indirettamente:

- **ProjectRepositoryCloner**:
  – Se la lista dei file modificati è ricavata tramite comandi Git, potrebbe essere necessario aggiornare la logica di clonazione/checkout.

Ne risulta il candidato set complessivo:

$$CIS = SIS \cup \{\texttt{ProjectRepositoryCloner}\}$$

4.2 Implementazione delle Modifiche

4.2.1 Actual Impact Set (AIS)

Le classi effettivamente modificate per implementare la modalità *Quick Scan* sono:

- `ProjectAnalyzer` (`components`):
  Esteso per supportare l'analisi su una lista limitata di file recenti, individuati tramite Git.
- `CodeSmileCLI` (`cli`):
  Modificato per gestire l'opzione `--quick-scan` da linea di comando.
- `ProjectRepositoryCloner` (`components`):
  Adattato per supportare l'estrazione dei file modificati nell'ultimo commit Git.

$$AIS = \{\texttt{ProjectAnalyzer}, \texttt{CodeSmileCLI}, \texttt{ProjectRepositoryCloner}\}$$

4.2.2 False Positive Impact Set

Nessuno degli elementi inizialmente ipotizzati nel *CIS* è risultato non modificato, dunque:

$$FPIS = \{\}$$

4.2.3 Discovered Impact Set

Nessuna classe ulteriore è stata coinvolta durante l'implementazione:

$$DIS = \{\}$$

4.3 Metriche

Una volta nota la composizione dell'AIS, possiamo calcolare le metriche di valutazione dell'analisi di impatto:

$$\text{Precision} = \frac{|CIS \cap AIS|}{|CIS|} = \frac{3}{3} = 1 \qquad \text{Recall} = \frac{|CIS \cap AIS|}{|AIS|} = \frac{3}{3} = 1$$

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)

5 Conclusioni

Il presente documento ha illustrato l'analisi di impatto relativa a tre Change Request implementate sul sistema **CodeSmile**.

Le analisi effettuate hanno mostrato:

- Una buona **precisione** nel prevedere le classi effettivamente modificate, soprattutto in CR3.
- Una **copertura completa** dell'impatto (precision e recall pari a 1.0) nella CR3, grazie a una chiara segmentazione dei componenti.
- La necessità di raffinare ulteriormente le previsioni per evitare falsi positivi (come osservato in CR1).
- L'introduzione di funzionalità esterne (come in CR2) può avvenire senza impattare l'architettura esistente, mantenendo l'indipendenza modulare del sistema.

Il lavoro ha inoltre confermato il ruolo centrale di alcune classi chiave, come `ProjectAnalyzer` e i servizi della Web App, evidenziando l'importanza di una struttura ben modulata per facilitare evoluzioni incrementali.

In sintesi, il sistema ha dimostrato una buona capacità di adattamento ai cambiamenti, mantenendo coerenza architetturale e manutenibilità.