UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

PRE-MODIFICATION TEST DOCUMENT

Ingegneria del Software: Tecniche Avanzate - Pre-Modification Test Document

Luigi Auriemma

Matricola: NF22500161

Ivan Chiello

Matricola: NF22500167

Repository: `https://github.com/LuigiAuriemma/smell_ai.git`

Anno Accademico 2025-2026


sesa
lab
SOFTWARE ENGINEERING
SALERNO

Indice

Elenco delle tabelle

CAPITOLO 1

Introduzione

1.1 Scopo del Documento

Lo scopo del presente documento è quello di presentare l'insieme dei casi di test designati a realizzare il testing di sistema del tool **CodeSmile** nella sua versione attuale (Baseline), prima dell'introduzione delle modifiche evolutive previste.
La strategia adottata è quella del **testing black-box**. Questo approccio consente di verificare la conformità del sistema rispetto ai requisiti funzionali concentrandosi esclusivamente sul comportamento esterno, ovvero analizzando la corrispondenza tra gli input forniti (directory e file sorgente) e gli output generati (report di analisi), senza ispezionare la logica interna del codice sorgente.

1.2 Ambiente di Testing

Al fine di garantire la coerenza e la riproducibilità dei risultati, i test di sistema vengono eseguiti in un ambiente controllato configurato come segue, in conformità con il file `requirements.txt` e la documentazione di progetto:

- **Interprete:** Python 3.11 (Runtime environment richiesto).

- **Dipendenze:** Installate tramite requirements.txt
- **Interfaccia di Input:** CLI .
- **Sistema Operativo di Riferimento:** Windows 11

1.3 Tipologia di Testing

Il metodo formale adottato per la progettazione dei casi di test è il **Category Partitioning** (Partizione delle Categorie).
Questa tecnica è stata utilizzata per scomporre il dominio degli input del tool in unità funzionali distinte (categorie), identificando per ciascuna di esse i parametri significativi e le relative classi di equivalenza. Tale approccio ha permesso di massimizzare la copertura funzionale dei Code Smell rilevabili (Generici e AI-Specific) minimizzando la ridondanza dei test.

CAPITOLO 2

Identificazione delle Categorie e dei Parametri

Il criterio adottato è quello del *category partitioning*, allo scopo di individuare le categorie di input e i parametri a cui associare ciascuna di esse. Di seguito sono elencati i gruppi di parametri definiti per il testing di sistema di CodeSmile, suddivisi per ambito logico: Input (File/Progetti), Code Smells (Oggetto dell'analisi) e Configurazione (Esecuzione).

2.1 Parametri File e Progetto

Fanno riferimento agli aspetti strutturali dei file sorgente e delle directory forniti in input al tool tramite la CLI.

Categoria	Codice	Scelte (Classi di Equivalenza)
Numero di File Input (NF) Cardinalità dei file sottoposti ad analisi all'interno della directory target.	NF0 NF1 NF2	Nessun file (Cartella vuota) Un singolo file Più file
Estensione File (EF) Tipologia di file presenti nel path di input.	EF0 EF1	Solo file supportati (`.py`) File misti o non supportati (es. `.c`, `.txt`)
Numero Progetti (NP) Modalità di scansione della root directory (singolo progetto o workspace multiplo).	NP0 NP1	Singolo Progetto Multi-Progetto (Flag `-multiple`)
Struttura Progetto (SP) Topologia delle directory.	SP0 SP1	Flat (Tutti i file nella root) Nested (Sottocartelle ricorsive)

Tabella 2.1: Parametri relativi ai File e alla Struttura

2.2 Parametri Code Smells

Fanno riferimento alla natura e alla quantità dei difetti (Smells) presenti nel codice sorgente analizzato.

Categoria	Codice	Scelte (Classi di Equivalenza)
Numero Code Smells (NCS) Quantità di smell iniettati o presenti nel codice target.	NCS0 NCS1 NCS2	Nessuno (Clean Code) Un singolo Code Smell Più Code Smells
Tipologia Code Smells (TCS) Classificazione dello smell in base alle regole del *Rule Checker*.	TCS0 TCS1 TCS2 TCS3	Nessuno Generic Smells (Pandas, Best Practices) AI-Specific Smells (PyTorch, TF) Misto (Generic + AI)

Tabella 2.2: Parametri relativi ai Code Smells

2.3 Parametri di Configurazione ed Esecuzione

Fanno riferimento alle opzioni operative della CLI (`cli_runner.py`) e all'esito dell'elaborazione.

Categoria	Codice	Scelte (Classi di Equivalenza)
Modalità Esecuzione (ME) Tipologia di elaborazione (sequenziale o parallela).	ME0 ME1	Sequenziale (Default) Parallela (Flag `-parallel`)
Numero Walkers (NW) Numero di worker process utilizzati (rilevante solo per ME1).	NW0 NW1	Default / Non specificato Specificati ($>$ 1 tramite `-max_walkers`)
Resume Analysis (RES) Ripresa di un'analisi interrotta.	RES0 RES1	False (Nuova analisi) True (Flag `-resume`)
Output Generation (OUT) Verifica della creazione del report finale.	OUT0 OUT1	Report non generato / Errore Report generato (`overview.csv`)

Tabella 2.3: Parametri di Configurazione ed Esecuzione

CAPITOLO 3

Test di Sistema

In questo capitolo vengono dettagliati i casi di test (Test Cases) progettati per coprire i frame identificati. La tabella seguente mappa ogni Test Case (ID) con il relativo **Test Frame** (combinazione di parametri) e l'**Oracolo** (risultato atteso).

TC ID	**Test Frame (Parametri)**	**Oracolo / Risultato Atteso**
TC_01	NF0, EF0, NP0, SP0, NCS0, TCS0, ME0, OUT0	**Empty Input:** L'esecuzione su directory vuota o input nullo termina correttamente gestendo l'eccezione (Exit 0) senza generare report.
TC_02	NF2, EF0, NP0, SP1, NCS1, TCS2, ME0, OUT1	**Nested AI-Smell:** Rilevamento di `DataFrame Conversion Misused` e `Chain Indexing` in sottocartelle annidate.
TC_03	NF1, EF0, NP0, SP0, NCS1, TCS2, ME0, OUT1	**Flat AI-Smell:** Rilevamento puntuale di `Matrix Multiplication API Misused` (np.dot) in file singolo.

TC ID	Test Frame (Parametri)	Oracolo / Risultato Atteso
TC_04	NF2, EF0, NP1, SP1, NCS2, TCS1, ME0, OUT1	**Multi-Project Generic:** Verifica smell generici (`Columns Dtype`, `In-Place`) su struttura multi-progetto complessa.
TC_05	NF1, EF0, NP0, SP0, NCS1, TCS1, ME1, OUT1	**Parallel Execution:** Verifica che l'analisi di un file con smell generici produca lo stesso output anche con flag `-parallel`.
TC_06	NF2, EF0, NP0, SP1, NCS0, TCS0, ME0, OUT1	**Clean Code Nested:** Analisi su file python puliti (`mocksum`, `checkeven`). Il report è generato ma vuoto (0 smell).
TC_07	NF1, EF0, NP0, SP0, NCS2, TCS2, ME0, OUT1	**PyTorch Specific:** Rilevamento simultaneo di `Chain Indexing` e `Gradients Not Cleared`.
TC_08	NF2, EF0, NP1, SP1, NCS0, TCS0, ME0, OUT1	**Clean Multi-Project:** Verifica assenza falsi positivi analizzando più progetti puliti contemporaneamente.
TC_09	NF1, EF0, NP0, SP0, NCS1, TCS2, ME0, OUT1	**Variant AI-Smell:** Rilevamento variante sintattica di `Matrix Multiplication API Misused`.
TC_10	NF1, EF0, NP0, SP0, NCS2, TCS3, ME0, OUT1	**Smoke Test:** Analisi base su file singolo con smell misti (Generici + AI).
TC_11	NF2, EF0, NP0, SP1, NCS2, TCS2, ME0, OUT1	**Deep Nested Stress:** Struttura profonda (Project1..4). Rilevamento multiplo di `Chain Indexing` e `Conversion Misused`.

TC ID	Test Frame (Parametri)	Oracolo / Risultato Atteso
TC_12	NF1, EF0, NP0, SP1, NCS2, TCS3, ME0, OUT1	**Mixed Context:** Rilevamento misto (`Columns Dtype` e `Chain Indexing`) in directory mockata.
TC_13	NF2, EF0, NP1, SP1, NCS2, TCS1, ME0, OUT1	**Report Separation:** Separazione corretta dei report tra `Project1` (Merge Param) e `Project2` (Chain Indexing).
TC_14	NF2, EF0, NP0, SP1, NCS1, TCS2, ME0, OUT1	**Noise Filtering:** Verifica di `Chain Indexing` ignorando file ausiliari puliti (`mocksum`) nella stessa cartella.
TC_15	NF1, EF1, NP0, SP0, NCS0, TCS0, ME0, OUT0	**Extension Ignored (.c):** Input contiene solo file C. Il sistema termina con successo (Exit 0) e nessun report.
TC_16	NF2, EF0, NP0, SP1, NCS2, TCS1, ME0, OUT1	**Generic Deep:** Rilevamento puntuale di smell generici (`In-Place`, `Deterministic Algo`) in struttura annidata.
TC_17	NF2, EF0, NP1, SP1, NCS2, TCS3, ME0, OUT1	**Complex Workspace:** Rileva `Merge Param` e `Chain Indexing` su due progetti distinti con sottocartelle.
TC_18	NF2, EF0, NP1, SP1, NCS0, TCS0, ME0, OUT1	**Clean Workspace:** Verifica assenza falsi positivi su struttura complessa con più progetti puliti.
TC_19	NF1, EF0, NP0, SP0, NCS2, TCS1, ME0, OUT1	**Single File Dense:** Rilevamento di `Empty Column` e `Columns Dtype` nello stesso file.

TC ID	Test Frame (Parametri)	Oracolo / Risultato Atteso
TC_20	NF1, EF0, NP0, SP0, NCS1, TCS2, ME0, OUT1	**Base Case AI:** Caso isolato per la verifica di `Chain Indexing` (AI-Specific).
TC_21	NF2, EF1, NP1, SP1, NCS-, TCS-, ME0, OUT1	**Robustness Mixed:** Il tool analizza i file `.py` validi ignorando i file `.c` e `.txt` sparsi nelle cartelle.