UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA

Ingegneria del Software: Tecniche Avanzate

Progetto ISTA – Evoluzione di CodeSmile

Pre-Modification System Testing

Professore:
Ch.mo Prof.
Andrea De Lucia

Tutor di riferimento:
Giammaria Giordano
Gilberto Recupito

Studenti:
Antonio D'Auria
Mat. NF22500063
Emanuele D'Auria
Mat. 0522501918
Luca Casillo
Mat. NF22500037

ANNO ACCADEMICO 2025/2026

Indice

Elenco delle tabelle

1 Introduzione e Scopo del Documento

Il presente documento descrive le attività di *Pre-Modification System Testing* condotte sul sistema CodeSmile, con l'obiettivo di definire e documentare il piano di test della versione corrente del sistema prima dell'applicazione delle Change Request.

In particolare, il documento stabilisce:

- la strategia di testing adottata;
- i livelli di testing considerati;
- i criteri di adeguatezza;
- i tool e framework utilizzati;
- l'approccio di testing di regressione.

Le informazioni riportate costituiscono il riferimento per il confronto con le attività di testing post-modifica.

2 Sistema sotto Test e Ambito

Il sistema sotto test corrisponde alla versione corrente di CodeSmile, come descritta nel documento di Reverse Engineering.

2.1 Ambito del Testing

Le attività di testing pre-modifica sono focalizzate esclusivamente sulle componenti del sistema interessate dagli interventi previsti. In particolare, il perimetro di analisi comprende:

- lo **Static Analysis Tool**, comprensivo delle modalità di esecuzione tramite interfaccia a riga di comando (CLI) e interfaccia grafica (GUI);
- la **Web Application**, includendo frontend, gateway e servizi backend.

Il componente *AI-Based Detection Tool* è da considerarsi **fuori scope** rispetto alle attività di testing documentate. Tale componente, pur essendo parte della suite CodeSmile, non è interessato dalle modifiche previste e non risulta integrato nei flussi di analisi coperti dalla test suite di sistema attuale.

3 Livelli di Testing e Organizzazione della Test Suite

La test suite di CodeSmile è organizzata secondo più livelli di testing, ciascuno caratterizzato da obiettivi e granularità differenti. Questa organizzazione riflette la struttura architetturale del sistema e consente di verificare il corretto funzionamento delle singole componenti, nonché delle loro interazioni.

3.1 Static Analysis Tool

I test relativi allo Static Analysis Tool sono collocati nella directory `test/` e risultano organizzati nei seguenti livelli:

- **Unit Testing** (`test/unit_testing`): verifica dei singoli package e moduli in isolamento, quali `cli`, `code_extractor`, `detection_rules` e `report`. I test unitari mirano a validare il comportamento delle singole funzioni e classi mediante input controllati.
- **Integration Testing** (`test/integration_testing`): verifica dell'interazione tra moduli cooperanti dello Static Analysis Tool, con particolare attenzione al corretto scambio di informazioni tra le componenti del core di analisi.
- **System Testing** (`test/system_testing`): validazione del comportamento complessivo dello Static Analysis Tool attraverso casi di test strutturati (TC1–TC21), basati su progetti di input realistici e directory di test.

3.2 Web Application

La Web Application adotta una strategia di testing multi-livello, coerente con la sua architettura a servizi e con la separazione tra frontend e backend.

- **Unit Testing Frontend** (`webapp/_tests_`): verifica dei componenti dell'interfaccia utente mediante Jest e Testing Library, con focus sul corretto rendering e sul comportamento delle singole componenti React.
- **Integration Testing** (`webapp/integration_tests`): verifica dell'interazione tra gateway e servizi backend, assicurando il corretto instradamento delle richieste e la coerenza delle risposte.
- **End-to-End Testing** (`webapp/cypress`): validazione dei principali flussi utente attraverso l'interfaccia web, simulando l'utilizzo reale del sistema.

4 Strategia di Testing

La strategia di testing adottata per la versione corrente di CodeSmile si basa su un approccio sistematico e multi-livello, finalizzato a garantire una copertura adeguata dei comportamenti rilevanti del sistema. L'obiettivo è individuare eventuali anomalie funzionali e strutturali prima dell'introduzione di modifiche evolutive.

4.1 Strategie di Testing Adottate

La strategia di testing di CodeSmile combina approcci di *white-box* e *black-box testing*, applicati in modo differenziato in base al livello di test e alla componente analizzata.

Per lo *Static Analysis Tool*, i test unitari adottano un approccio *white-box*, in quanto verificano il comportamento interno di moduli che operano sull'Abstract Syntax Tree, sull'estrazione delle informazioni dal codice sorgente e sull'applicazione delle singole detection rules.

I test di integrazione verificano il corretto coordinamento tra moduli cooperanti del core di analisi e adottano un approccio misto, basato sulla conoscenza delle interfacce tra le componenti ma non dei dettagli interni delle singole implementazioni.

I test di sistema adottano un approccio *black-box*, validando il comportamento complessivo dello Static Analysis Tool esclusivamente in termini di input forniti (progetti o directory di codice) e output prodotti.

Per la *Web Application*, i test unitari frontend adottano un approccio *white-box* limitato ai componenti React, mentre i test di integrazione e i test end-to-end sono di tipo *black-box*, in quanto verificano i flussi applicativi dal punto di vista dell'utente finale.

4.2 Strumenti di Testing

Le attività di testing di CodeSmile si basano su strumenti consolidati, selezionati in funzione delle tecnologie adottate dal sistema.

Per lo Static Analysis Tool, i test unitari, di integrazione e di sistema sono implementati mediante `pytest`.

Il testing frontend della Web Application utilizza `Jest` e `React Testing Library` per la verifica dei componenti React, mentre i test di integrazione backend impiegano strumenti di testing Python analoghi a quelli dello Static Analysis Tool.

I test end-to-end della Web Application sono implementati tramite `Cypress`, che consente di validare i flussi completi dell'applicazione simulando l'interazione dell'utente con l'interfaccia web.

4.3 Strategia di Regression Testing

La strategia di *regression testing* adottata per CodeSmile è finalizzata a garantire che le funzionalità esistenti continuino a operare correttamente a seguito dell'introduzione di modifiche evolutive o correttive al sistema.

Considerata la natura modulare dell'architettura e la presenza di una test suite già strutturata su più livelli, viene adottato un approccio di **regression testing selettivo**, in cui l'esecuzione dei test è guidata dall'analisi delle componenti impattate dalle modifiche.

In particolare:

- i **test unitari** vengono rieseguiti per tutti i moduli direttamente modificati e per quelli che dipendono dalle interfacce coinvolte;
- i **test di integrazione** vengono eseguiti per verificare che le interazioni tra componenti cooperanti non siano state alterate;
- i **test di sistema** vengono utilizzati come test di non-regressione per validare il comportamento complessivo del sistema rispetto a scenari realistici di utilizzo.

Per lo *Static Analysis Tool*, la regressione si concentra sulla corretta rilevazione dei machine learning-specific code smells e sulla stabilità del flusso di analisi, incluse le modalità di esecuzione tramite CLI e GUI.

Per la *Web Application*, la regressione è verificata mediante test di integrazione e test end-to-end, assicurando che i principali flussi utente (caricamento dei progetti, avvio dell'analisi, generazione dei report) rimangano invariati dal punto di vista funzionale.

4.4 Category Partition e Weak Equivalence Class Testing

Per la progettazione dei test è stata adottata una strategia basata sulla combinazione del *Category Partition Method* e del *Weak Equivalence Class Testing*.

Il Category Partition Method è utilizzato per identificare in modo sistematico i parametri rilevanti di ciascuna funzionalità, definire le categorie associate e partizionare ciascuna categoria in scelte valide e invalide. Questo approccio consente di esplicitare lo spazio dei test in maniera chiara e strutturata.

Il criterio di Weak Equivalence Class Testing è applicato nella fase di selezione dei test frame, al fine di limitare il numero di combinazioni da testare. In particolare, per ciascuna funzionalità viene definito:

- un test frame valido, contenente esclusivamente scelte valide;
- un insieme di test frame invalidi, ciascuno contenente una sola scelta invalida, mantenendo valide tutte le altre.

La combinazione dei due approcci consente di ottenere una copertura significativa dei comportamenti rilevanti del sistema evitando l'esplosione combinatoria del numero di test.

4.4.1 Static Analysis Tool. L'applicazione del Category Partition Method allo *Static Analysis Tool* considera il flusso di analisi nel suo complesso, indipendentemente dalla modalità di avvio (CLI o GUI), in quanto entrambe convergono sul medesimo core di analisi.

La Tabella 4.1 riporta le categorie individuate, le relative scelte (identificate da un codice univoco) e la loro validità. Eventuali dipendenze o vincoli tra scelte sono indicati tra parentesi quadre.

Tabella 4.1. Category Partition - Static Analysis Tool

Categoria	Scelta	Validità
Tipo di Input (TI)	TI.1 – Singolo file Python	Valida
	TI.2 – Directory contenente almeno un file Python	Valida
	TI.3 – Directory contenente più progetti	Valida
Validità del Path di Input (VP)	VP.1 – Path esistente e accessibile	Valida
	VP.2 – Path inesistente o non accessibile [ERROR]	Invalida
Modalità di Analisi (MA)	MA.1 – Analisi di un singolo progetto	Valida
	MA.2 – Analisi multiprogetto	Valida
Esecuzione Parallela (EP)	EP.1 – Esecuzione sequenziale	Valida
	EP.2 – Esecuzione parallela [MA.2]	Valida
Numero di Walkers (NW)	NW.1 – Valore ≤ 0 [EP.2] [ERROR]	Invalida
	NW.2 – Valore < 5 [EP.2]	Valida
	NW.3 – Valore $= 5$ (default) [EP.2]	Valida
	NW.4 – Valore > 5 [EP.2]	Valida
Resume (RES)	RES.1 – Resume disattivato	Valida
	RES.2 – Resume abilitato [MA.2, EP.1]	Valida
Presenza di File Python (PF)	PF.1 – Almeno un file Python rilevato	Valida
	PF.2 – Nessun file Python rilevato [TI.2]	Valida
Output Generato (OUT)	OUT.1 – Generazione file `overview.csv`	Valida
	OUT.2 – Generazione file di dettaglio per progetto [MA.2]	Valida

Vincoli tra Scelte. Le categorie e le scelte individuate non sono indipendenti tra loro. Le principali dipendenze funzionali sono le seguenti.

- Le scelte relative all'analisi multiprogetto sono ammissibili solo quando l'input rappresenta più progetti:
  - **TI.3** $\implies$ **MA.2**;
  - **EP.2** $\implies$ **MA.2**;
  - **OUT.2** è ammissibile solo con **MA.2**.
- La scelta **RES.2** è significativa solo per analisi multiprogetto in modalità sequenziale: **RES.2** $\implies$ **MA.2** $\wedge$ **EP.1**
- Nel caso di input costituito da singolo file Python (**TI.1**), la presenza di file Python è implicita e la categoria **PF** non è applicabile.

- La categoria **Numero di Walkers (NW)** è significativa esclusivamente in presenza di esecuzione parallela:
  - tutte le scelte **NW.*** sono ammissibili solo se **EP.2** è selezionata;
  - la scelta **NW.1** comporta una terminazione con errore per validazione degli argomenti.
- La validità del path di input è un prerequisito per l'avvio dell'analisi: la selezione di **VP.2** comporta la terminazione immediata dell'esecuzione.
- La condizione **PF.2** (assenza di file Python) genera errore nel caso di analisi su singolo progetto (**MA.1**) con input directory (**TI.2**). Nel caso di analisi multiprogetto (**MA.2**), l'analisi può completarsi senza produrre risultati per i sottoprogetti privi di file Python.

Derivazione dei Test Case. I test case di sistema sono derivati secondo il criterio di *Weak Equivalence Class Testing*, selezionando:

- test case validi rappresentativi dei principali flussi di esecuzione;
- test case invalidi, ciascuno caratterizzato da una singola scelta non valida, mantenendo valide tutte le altre.

Tabella 4.2. Test Case Mapping - Static Analysis Tool

Test Case ID	Scelte Rappresentative	Oracolo Atteso
TC_01	TI.2, VP.1, MA.1, EP.1, PF.1, OUT.1	Analisi completata; se vengono rilevati code smell, viene generato il file `overview.csv`.
TC_02	VP.2	Errore segnalato: path di input non valido.
TC_03	TI.2, VP.1, MA.1, PF.2	Errore segnalato: nessun file Python da analizzare.
TC_04	TI.3, VP.1, MA.2, EP.2, NW.3, OUT.2	Analisi multiprogetto completata; per i progetti con risultati vengono generati i file di dettaglio e, se presenti risultati complessivi, il file `overview.csv`.
TC_05	TI.3, VP.1, MA.2, EP.2, NW.1	Errore segnalato: numero di walkers non valido.
TC_06	TI.3, VP.1, MA.2, EP.2, NW.3	Analisi completata senza risultati; non viene generato il file `overview.csv`.
TC_07	TI.1, VP.1, MA.1, EP.1, OUT.1	Analisi completata sul singolo file; se vengono rilevati code smell, viene generato il file `overview.csv`.
TC_08	TI.3, VP.1, MA.2, EP.1, RES.2	Esecuzione con resume: l'analisi riprende dal file `execution_log.txt`.

4.4.2 Web Application. L'applicazione del Category Partition Method alla *Web Application* considera i principali flussi utente e le validazioni funzionali implementate lato frontend e gateway, con particolare attenzione ai punti di interazione tra utente e sistema.

La Tabella 4.3 riporta le categorie individuate, le scelte associate e la loro validità. Eventuali dipendenze tra scelte sono indicate tra parentesi quadre.

Tabella 4.3. Category Partition - Web Application

Categoria	Scelta	Validità
Flusso Utente (FU)	FU.1 – Navigazione Home	Valida
	FU.2 – Analisi singolo file Python	Valida
	FU.3 – Analisi progetto/i	Valida
	FU.4 – Generazione report	Valida
Modalità di Analisi (AM)	AM.1 – Analisi AI [FU.2/FU.3]	Valida
	AM.2 – Analisi Static Tool [FU.2/FU.3]	Valida
Selezione File Python (SF)	SF.1 – File Python valido selezionato [FU.2]	Valida
	SF.2 – Nessun file selezionato [FU.2] [ERROR]	Invalida
	SF.3 – File selezionato non Python [FU.2] [ERROR]	Invalida
Gestione Progetti in Context (PC)	PC.1 – Nessun progetto presente [FU.3/FU.4] [ERROR]	Invalida
	PC.2 – Almeno un progetto presente [FU.3/FU.4]	Valida
File di Progetto (FP)	FP.1 – Progetto con almeno un file Python [FU.3, PC.2]	Valida
	FP.2 – Progetto senza file Python [FU.3, PC.2] [ERROR]	Invalida
Esito Generazione Report (RR)	RR.1 – Report generato correttamente [FU.4, PC.2]	Valida
	RR.2 – Errore in generazione report [FU.4, PC.2] [ERROR]	Invalida
Dati Report (DR)	DR.1 – Dati presenti [RR.1]	Valida
	DR.2 – Nessun dato disponibile [RR.1]	Valida

Vincoli tra Scelte. Le scelte individuate non sono indipendenti tra loro. I principali vincoli funzionali sono elencati di seguito.

- La categoria **Modalità di Analisi (AM)** è significativa esclusivamente nei flussi di analisi: **AM.**$* \implies$ **FU.2** $\vee$ **FU.3**
- Nel flusso **FU.2** (analisi singolo file Python), la selezione di un file valido è obbligatoria: **SF.2** $\vee$ **SF.3** $\implies$ errore e analisi non avviata
- I flussi **FU.3** (analisi progetto/i) e **FU.4** (generazione report) richiedono la presenza di almeno un progetto in context: **PC.1** $\implies$ operazione non eseguibile
- Nel flusso **FU.3**, ciascun progetto analizzato deve contenere almeno un file Python: **FP.2** $\implies$ errore per il progetto interessato

- La generazione del report (**FU.4**) è ammissibile solo se sono soddisfatti i prerequisiti funzionali: **RR.**$*$ $\implies$ **PC.2**
- In caso di report generato correttamente (**RR.1**), il sistema può produrre:
  - dati disponibili (**DR.1**);
  - nessun dato disponibile (**DR.2**), con visualizzazione di un messaggio informativo.

Derivazione dei Test Case. I test case di sistema sono derivati selezionando combinazioni rappresentative delle classi di equivalenza individuate. In accordo con il criterio di *Weak Equivalence Class Testing*:

- sono definiti test case validi che coprono i principali flussi utente;
- sono definiti test case invalidi, ciascuno caratterizzato da una sola scelta non valida, mantenendo valide le restanti e rispettando i vincoli.

La Tabella 4.4 riporta il mapping tra test case, scelte rappresentative e oracolo atteso.

Tabella 4.4. Test Case Mapping - Web Application

Test Case ID	Scelte Rappresentative	Oracolo Atteso
WTC_01	FU.1	Le principali pagine dell'applicazione sono correttamente raggiungibili tramite la navigazione.
WTC_02	FU.2, AM.1, SF.1	Analisi completata con visualizzazione dei risultati.
WTC_03	FU.2, SF.3	Errore segnalato: file selezionato non valido; analisi non avviata.
WTC_04	FU.3, PC.2, FP.1, AM.2	Analisi dei progetti completata con visualizzazione dei risultati.
WTC_05	FU.3, PC.2, FP.2	Errore segnalato: progetto privo di file Python.
WTC_06	FU.4, PC.1	Errore segnalato: nessun progetto disponibile per la generazione del report.
WTC_07	FU.4, PC.2, RR.1, DR.1	Report generato correttamente con visualizzazione del grafico.
WTC_08	FU.4, PC.2, RR.2	Errore segnalato durante la generazione del report.
WTC_09	FU.4, PC.2, RR.1, DR.2	Report generato senza dati; visualizzazione di messaggio informativo.