Post-Modification & Regression Testing Document

Versione 1.0

CodeSmile

Team Members:
Matteo Ercolino – 0522501462
Simone Silvestri – 0522501419

Repository: GitHub link

Anno Accademico 2024/2025

Indice

1 Introduzione

Il presente documento descrive le attività di testing svolte sul sistema **CodeSmile** in seguito all'introduzione delle Change Requests CR1, CR2 e CR3. L'obiettivo è duplice:

- Verificare il corretto funzionamento delle nuove funzionalità introdotte (**Post-Modification Testing**);
- Assicurarsi che il sistema preesistente non abbia subito regressioni funzionali (**Regression Testing**).

2 Post-Modification Testing

Le attività di Post-Modification Testing hanno incluso il 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:

2.1 CR1 – Dashboard Interattiva

Tipo di modifica: Perfettiva (Frontend + Backend)
Componenti coinvolte: `report_main.py`, `report_router.py`, `report_utility.py`, `app/reports/page.tsx`, `components/ChartContainer.tsx`, `components/PDFDownloadButton.tsx`, `components/ProjectSidebar.tsx`, `components/SmellDensityCard.tsx`

- Cypress E2E e test di sistema per il rendering della dashboard e la corretta aggregazione dei dati.

Esito: Verificata la piena funzionalità. Nessuna discrepanza rilevata.

2.2 CR2 – Integrazione CI/CD

Tipo di modifica: Adattiva (nuovo modulo)

- Testata l'esecuzione automatica in ambiente CI e la generazione delle issue tramite `CIExecutor` e `CIReportParser`.

Esito: Integrazione validata. Nessun errore riscontrato.

2.3 CR3 – Modalità `Quick Scan`

Tipo di modifica: Perfettiva (ottimizzazione CLI)

- Confrontati i risultati dell'analisi veloce con quelli della modalità completa: equivalenti.

Esito: Funzionalità conforme ai requisiti.

3 Regression Testing

3.1 Obiettivo

Verificare che l'introduzione delle Change Requests non abbia alterato il comportamento atteso del sistema esistente.

3.2 Strategia

- Riesecuzione completa della suite di test pre-esistenti: unità, integrazione, sistema.
- Confronto degli output generati con quelli attesi.
- Validazione automatica tramite pipeline GitHub Actions.

3.3 Esecuzione Test Suite

Test automatici

- **Unit test:** 166/166 superati
- **Integration test:** 8/8 superati
- **System test:** 21 scenari preesistenti eseguiti con successo
- **E2E WebApp (Cypress):** 20/20 scenari superati

Copertura

- **Branch coverage:** 87%
  - Il valore di coverage complessivo (linee e rami) è pari a 87%, con branch coverage attiva come da configurazione. I moduli core superano il 90%, con piena copertura (100%) per i rilevatori principali e i moduli CI/CD. La copertura parziale è limitata ad alcuni rami condizionali nelle componenti CLI e GUI.
- **Badge Codecov:** aggiornato correttamente

4 Dettaglio dei Test

In questo capitolo vengono riportati i principali casi di test introdotti o aggiornati per ciascuna Change Request, in modo da fornire tracciabilità puntuale tra requisito, componente e verifica eseguita.

4.1 Unit Test

4.1.1 CR1 – Dashboard Interattiva

Sono stati aggiunti test di unità per componenti UI frontend (React/Vue) tramite Jest e Cypress component testing. I test verificano il corretto rendering delle viste e la corretta ricezione dei dati dalle API REST. Esempi:

- Rendering condizionato dei grafici.
- Gestione di fallback e loading state.
- Interazione con componenti `metrics-summary`, `smells-table`.

4.1.2 CR2 – Integrazione CI/CD

ID	Funzione / Classe	Input	Oracolo atteso
UT-CR2-01	`CIExecutor.run_quick_scan`	Patch `subprocess.run` con successo	Output su stdout, nessuna eccezione
UT-CR2-02	`CIExecutor.run_quick_scan`	Patch `subprocess.run` con errore	Messaggio di errore stampato
UT-CR2-03	`CIReportParser.generate_summary`	CSV con smells	Markdown contenente smells per file
UT-CR2-04	`CIReportParser.generate_summary`	CSV vuoto	Markdown con messaggio “no smells”
UT-CR2-05	`CIReportParser.exists`	File presente o assente	`True` o `False`
UT-CR2-06	`CIReportParser.load`	CSV con entry	DataFrame equivalente a input
UT-CR2-07	`CIReportParser.to_markdown`	CSV vuoto	Stringa con testo ”No code smells found.”
UT-CR2-08	`CIReportParser.get_smell_count`	CSV con 2 entry	Valore 2
UT-CR2-09	`CIReportParser.generate_issue_metadata`	CSV con smell	Dizionario con chiavi `title`, `body`

4.1.3 CR3 – Modalità `Quick Scan`

ID	Funzione / Classe	Input	Oracolo atteso
UT-CR3-01	`CodeSmileCLI.execute`	Argomenti validi, mock `analyze_project`	Output di successo, analisi eseguita

UT-CR3-02	`CodeSmileCLI.execute`	Argomenti mancanti	Eccezione `SystemExit`, errore stampato
UT-CR3-03	`CodeSmileCLI.execute`	Parallel + `max_walkers = -1`	`ValueError` con messaggio atteso
UT-CR3-04	`CodeSmileCLI.execute`	Modalità `--parallel` + multiprogetto	Chiamate a `analyze_projects_parallel`, `merge_results`
UT-CR3-05	`CodeSmileCLI.execute`	Modalità sequenziale	Chiamata a `analyze_project`
UT-CR3-06	`CodeSmileCLI.execute`	Flag `--resume` attivo	`clean_output_directory` non chiamata
UT-CR3-07	`CodeSmileCLI.execute`	Quick scan attivo + commit depth	Chiamata a `analyze_recent_files`, output stampato
UT-CR3-08	`CodeSmileCLI.execute`	`max_walkers = 0`, parallel = True	`ValueError` lanciato
UT-CR3-09	`CodeSmileCLI.execute`	Print config abilitato	Tutti i parametri stampati correttamente
UT-CR3-10	`CodeSmileCLI.execute`	`resume=True`, multiprogetto	`clean_output` non chiamata, `merge` eseguito

4.2 Integration Test

Questa sezione descrive i test di integrazione introdotti per verificare il corretto funzionamento tra moduli o servizi distinti, con particolare attenzione al gateway WebApp, ai servizi CI/CD, e alla nuova modalità `quick scan`.

4.2.1 CR1 – Gateway → Report Service (WebApp)

I seguenti test validano l'integrazione tra l'API Gateway FastAPI e il microservizio di generazione report.

ID	Descrizione	Input	Esito atteso
IT-CR1-01	Invio di progetto valido con smells	JSON strutturato (1 progetto, 1 smell)	HTTP 200, struttura report completa

4.2.2 CR2 – CI/CD Integration

Questi test verificano la corretta invocazione degli script CLI da parte del modulo CIExecutor e la generazione del report da parte del parser.

ID	Descrizione	Input	Esito atteso
IT-CR2-01	Invocazione CLI da CIExecutor	Mock `subprocess.run` con args	Comandi corretti (input, output, quick-scan, commit-depth)
IT-CR2-02	Parser CSV completo	File CSV con 2 smells distinti	Markdown generato + metadati issue

4.2.3 CR3 – CLI (Quick Scan)

Il test di integrazione associato alla CR3 è un'estensione dei test esistenti sulla CLI. È stato aggiornato per includere e verificare il nuovo parametro `--quick-scan`, senza introdurre nuovi scenari.

ID	Descrizione	Input	Esito atteso
IT-CR3-01	Verifica flag `--quick-scan` attivo	CLI con argomento `--quick-scan`	Analisi eseguita su file recenti, log corretti

4.3 System Test

I test di sistema pre-esistenti (21 scenari) sono stati rieseguiti con successo. Di seguito i nuovi test case introdotti per validare la modalità `Quick Scan`, progettati secondo il *Category Partition Method*:

Test Case ID	Configurazione e Oracolo
TC_22	**Parametri:** NF = 1, EF = .py, NP = 1, SD = semplice, NCS = 2, TCS = API-specific, ME = CLI, EP = false, NW = n/a, ERR = nessun errore, RES = false, **quick_scan = true**, commit_depth = 1 **Oracolo**: Il tool esegue correttamente la modalità Quick Scan su file modificati nell'ultimo commit. Viene generato un file `quickscan_results.csv` contenente gli smells individuati.
TC_23	**Parametri:** NF = 0 (nessun file modificato), EF = .py, NP = 1, SD = semplice, NCS = 0, TCS = (n/a), ME = CLI, EP = false, NW = n/a, ERR = nessun errore, RES = false, **quick_scan = true**, commit_depth = 1 **Oracolo**: Nessun file viene analizzato perché non vi sono modifiche recenti. Il tool termina correttamente e mostra il messaggio "Quick Scan: 0 code smells trovati.". Nessun file `quickscan_results.csv` viene creato.
TC_24	**Parametri:** Repo Git privo dei branch `main` e `master`, quick_scan = true, commit_depth = 1 **Oracolo**: Il tool rileva che nessun branch standard è presente e termina con errore controllato o messaggio informativo, senza crash. Nessun file viene analizzato.
TC_25	**Parametri:** NF >1, EF = .py, NP = 1, SD = semplice, NCS >1, TCS = misto, ME = CLI, quick_scan = true, commit_depth = 3 **Oracolo**: Il tool esegue la Quick Scan su file modificati negli ultimi 3 commit, aggrega gli smells da commit diversi e genera `quickscan_results.csv` con metadati commit (autore, messaggio, hash).

Scenario CR2 – Verifica Pipeline CI/CD con `act`

Per la Change Request **CR2** è stato progettato un test di sistema che verifica il corretto funzionamento della pipeline CI/CD automatizzata. L'intera pipeline è stata eseguita localmente mediante `act`, uno strumento per la simulazione di GitHub Actions su ambienti locali.

Il test riproduce un flusso realistico in cui:

- il workflow GitHub Actions viene attivato automaticamente;
- la pipeline esegue il ci_executor.py;
- viene generato un report in formato CSV contenente i code smells;

- il modulo `CIReportParser` converte il report in markdown e metadati per GitHub Issues.

L'esito del test è stato verificato tramite:

- codice di uscita del job (success/failure);
- contenuto del file di output;
- presenza e formato del markdown generato nel log di `act`;
- validazione dei messaggi di console e creazione issue.

Questo test garantisce la correttezza del flusso end-to-end dalla modifica del codice alla generazione del risultato nella pipeline CI, senza l'intervento manuale su GitHub.

4.4 End-to-End Testing (WebApp)

Per la CR1 è stata eseguita una campagna di test End-to-End (E2E) sulla dashboard della WebApp, realizzata con React, utilizzando Cypress come framework di test. L'obiettivo è stato verificare che l'interfaccia grafica rispondesse correttamente ai principali flussi di interazione utente, con particolare attenzione alla visualizzazione dei report e alla robustezza in presenza di dati mancanti o incompleti.

I test E2E includono:

- **Rendering iniziale:** verifica della presenza del messaggio di assenza dati quando nessun progetto è caricato.
- **Visualizzazione dinamica del report:** test dell'interfaccia utente dopo l'inserimento di un progetto contenente smells. Viene verificata la corretta visualizzazione di grafici, tabelle e pannelli riassuntivi.
- **Funzionalità di export:** verifica del download del report in PDF quando sono disponibili i dati.
- **Gestione progetti senza smell:** controllo dell'assenza di grafici e messaggi d'errore nel caso in cui il progetto selezionato non contenga problemi rilevati.

5 Conclusioni

Tutte le attività di Post-Modification e Regression Testing sono state completate con successo. Le nuove funzionalità introdotte risultano integrate correttamente, mentre il comportamento del sistema preesistente è stato preservato. Non sono state identificate regressioni né malfunzionamenti.