Final Report

MARK 2.0 Plus

Data	24/01/2026
Presentato da	Cerchia Giovanni (NF22500202) Medica Vincenzo (NF22500203)
Repository	https://github.com/vmedica/MARK-2.0-Plus

Sommario

1 Panoramica

MARK 2.0 è un tool di analisi statica per repository Python nel dominio del Machine Learning, progettato per supportare attività di ricerca nel contesto SE4AI (Software Engineering for AI-based Systems). Il sistema consente la classificazione automatica di progetti ML come:

- **Producer**: progetti che addestrano modelli.
- **Consumer**: progetti che utilizzano modelli.

L'evoluzione **MARK 2.0 Plus** introduce tre estensioni tramite Change Request:

- **CR1:** integrazione di metriche quantitative di qualità del codice;
- **CR2:** interfaccia grafica per configurazione ed esecuzione;
- **CR3:** dashboard per reportistica aggregata.

1.1 Risultati chiave ottenuti

L'estensione MARK 2.0 Plus ha prodotto i seguenti risultati:

- **Integrazione strutturata delle metriche di qualità (CR1):** il core è stato esteso con il calcolo automatico di Cyclomatic Complexity (CC) e Maintainability Index (MI) tramite la libreria Radon, con aggregazione a livello di progetto.
- **Introduzione di un layer di configurazione (CR2):** la GUI sviluppata in Tkinter consente di configurare ed eseguire l'analisi senza la necessità di ricorrere alla CLI e di modificare manualmente i parametri all'interno del codice.
- **Dashboard grafica integrata (CR3):** i risultati sono presentati tramite grafici dinamici (matplotlib) che rendono immediata l'interpretazione dei risultati.
- **Elevata copertura di test:** le componenti modificate raggiungono una branch coverage complessiva per Unit ed Integration Testing del 95% nei test white-box post-modification.
- **Baseline verificata:** prima delle modifiche sono stati creati e validati due casi d'uso principali (Analysis e Cloning), da cui è stata derivata una suite di 16 system test riutilizzata per il regression testing.
- **Stabilità del baseline garantita:** tutti i test di sistema pre-modification sono stati rieseguiti con successo, confermando l'assenza di regressioni.

2 Pre-modification

Prima dell'introduzione delle Change Requests, è stata definita e validata una suite di System Testing, con l'obiettivo di costruire una base di riferimento riutilizzabile per il Regression Testing post-modification.

2.1 Casi d'uso testati

Per la creazione del System Testing in modo più efficace e controllabile, l'utilizzo del sistema è stato scomposto in due funzionalità/casi d'uso separati:

- **UC-1 (Analysis)**: analisi/classificazione di una directory locale contenente repository;
- **UC-2 (Cloning)**: clonazione di repository Git da una lista CSV.

I casi di test sono stati progettati mediante un **approccio black-box funzionale**, utilizzando la tecnica di Category Partition, che ha portato alla creazione di 16 test case:

- **UC-1 (Analysis):** 12 test case;
- **UC-2 (Cloning):** 4 test case.

Tali test sono stati eseguiti con successo, senza alcun fallimento.
La suite di 16 system test ha costituito la baseline che è stata successivamente riutilizzata come suite di regression testing, per verificare che le modifiche introdotte non abbiano compromesso il comportamento corretto del tool.

3 Change Requests Implementate

Dopo aver definito la suite di System Testing, si è previsto di estendere MARK 2.0 attraverso le **Change Requests (CR)** illustrate di seguito.

ID	Descrizione	Tipo	Stato
CR1	Metriche CC e MI tramite Radon	Enhancement (Additive + Perfective)	Rilasciata
CR2	GUI per configurazione ed esecuzione	Enhancement (Additive)	Rilasciata
CR3	Dashboard grafica	Enhancement (Additive + Perfective)	Rilasciata

4 Impact Analysis

Prima dell'implementazione delle Change Requests (CR) è stata definita l'Impact Analysis riportata nella tabella sottostante, che ha previsto l'analisi dell'impatto su MARK 2.0 per le tre CR, con l'obiettivo di identificare e valutare le componenti interessate dalle modifiche introdotte.

ID	Classi	Metriche	Impatto
CR1	|SIS| = 3 |CIS| = 13 |AIS| = 10 |FPIS| = 3 |DIS| = 0	Precision = 0,77 Recall = 1	Alto
CR2	|SIS| = 0 |CIS| = 0 |AIS| = 0 |FPIS| = 0 |DIS| = 0	Dato che **SIS =** ⌀ e **CIS =** ⌀, le metriche (Precision/Recall) non risultano informative.	Basso
CR3	|SIS| = 0 |CIS| = 0 |AIS| = 0 |FPIS| = 0 |DIS| = 0	Dato che **SIS =** ⌀ e **CIS =** ⌀, le metriche (Precision/Recall) non risultano informative.	Basso

5 Post Modification Testing

Dopo l'implementazione delle Change Requests (CR) sulle componenti interessate dalle modifiche è stato svolto:

- **Basic Unit Testing (white-box)**: focalizzato sui metodi, con mocking dove necessario per isolare dipendenze (file system/servizi).
- **Integration Testing (white-box)**: sugli stessi metodi ma senza mocking, verificando la corretta interazione con il sistema.

La selezione degli input è stata guidata dalla costruzione del CFG dei metodi target (script Python basato su AST), scegliendo percorsi che coprono il maggior numero di branch.
Di seguito sono illustrati i risultati ottenuti per lo Unit e l'Integration Testing.

5.1 Risultati per lo Unit Testing

Package/Classe	Branch coverage	Note
`modules/analyzer/MLAnalyzer`	98%	CR1
`gui/services/pipeline_service.py`	100%	CR2
`gui/services/output_reader.py`	93%	CR2/CR3
Totale	**97%**	

5.2 Risultati per l'Integration Testing

Package/Classe	Branch coverage	Note
`modules/analyzer/MLAnalyzer`	100%	CR1
`gui/services/pipeline_service.py`	100%	CR2
`gui/services/output_reader.py`	93%	CR2/CR3
`gui/controller.py`	85%	CR2/CR3
Totale	**95%**	

5.3 Post-Modification System Testing

Dopo aver svolto lo Unit e l'Integration Testing si è condotto il system testing in modalità **black-box**, progettando i casi di test tramite **Category Partition** che prevede: l'identificazione di parametri/oggetti dell'ambiente, definizione di categorie e scelte, costruzione di test frame validi e derivazione dei test case con oracoli.

6 Regression Testing

Al termine del System Testing, il Regression Testing è stato eseguito tramite la riesecuzione completa della suite di test preesistenti, con particolare riferimento ai test di sistema eseguiti nella fase pre-modification.
Nel nostro caso, la suite era composta da **16 test case**, definiti sui due casi d'uso principali (Analysis 12 test case e Cloning 4 test case) e completati senza fallimenti in fase pre-modification.

Totale	Superati	Falliti
16	16	0

7 Sintesi dei Risultati del Post-Modification e Regression Testing

Le attività di Post-Modification Testing e Regression Testing hanno permesso di:

- verificare con test white-box (unit e integration) i metodi selezionati, guidati dai CFG, raggiungendo una branch coverage pari a **95%** sulle classi target;
- validare a livello di sistema le funzionalità introdotte con **CR1–CR3** tramite Category Partition (UC-CR1… UC-CR3);
- confermare l'assenza di regressioni mediante la riesecuzione della suite di pre-modifica (16 test case).

7.1 Dati finali

Al termine dell'esecuzione dei vari test, tutti gli 84 casi di test previsti sono stati eseguiti con successo, senza riscontrare alcun fallimento.
Risultati finali:

- Test eseguiti: 84
- Fallimenti: 0
- Incident / anomalie rilevate: nessuna

Fase	Tipo	Riferimento	Numero
Test eseguiti post-modification	**Unit Test**	CR1	8
		CR2	8
		CR3	2
	Integration Test	CR1	8
		CR2	13
		CR3	7
	System Test (CR1, CR2, CR3)	CR1	6
		CR2	12
		CR3	4
Regression Testing	**System Test Pre Modification**	UC-1 Analysis	12
		UC-2 Cloning	4
Totale			**84**

8 Conclusioni e Sviluppi Futuri

Il progetto ha consolidato un processo di evoluzione controllata basato su change management, impact analysis e testing sistematico.
Le tre Change Requests hanno aumentato significativamente l'osservabilità del sistema, l'usabilità e il valore informativo dei risultati, mantenendo al contempo la stabilità del baseline.
Come sviluppi futuri si prevede di:

- Estendere il numero di tipologie di classificazione ML (oltre Producer/Consumer).
- Integrare nuove metriche di qualità (es. coupling, cohesion).
- Arricchire la GUI con nuovi grafici e filtri interattivi.