Pre-Modification System Testing

MARK 2.0 Plus

Data	23/01/2026
Presentato da	Cerchia Giovanni (NF22500202) Medica Vincenzo (NF22500203)
Repository	https://github.com/vmedica/MARK-2.0-Plus

Sommario

1 Introduzione

Lo scopo del presente documento è descrivere la **suite di System Testing** svolta sul sistema **prima** dell'applicazione di qualunque Change Request, così da definire una baseline riutilizzabile per il **Regression Testing post-modifica**.

1.1 Scomposizione in casi d'uso

Per eseguire il 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.

Nota: l'analisi **(UC-1)** può essere eseguita **anche senza cloning**, a condizione di disporre di una directory con repository già presenti localmente.

1.2 Entry-point per l'esecuzione dei test: `main_args.py`

Allo stato iniziale, l'avvio era disponibile solo tramite `main.py`, ma diversi parametri risultavano configurati tramite variabili hard-coded, rendendo meno pratico eseguire più scenari in modo ripetibile senza modifiche manuali al codice.

Per questo motivo, per l'attività di system testing è stata introdotta una variante dell'entry-point, denominata `main_args.py`, invocabile da terminale con **CLI args**, così da selezionare i parametri necessari ai test mantenendo invariato il codice sorgente.

1.3 Tipologia e tecnica di testing

La tecnica scelta è il **testing funzionale/black-box**, che consente di verificare il comportamento esterno del sistema rispetto ai requisiti specificati, senza analizzarne l'implementazione interna.

Per la progettazione dei casi di test è stato adottato il metodo **Category Partition**, una strategia che permette di:

- individuare i **parametri** di input e gli **oggetti dell'ambiente** rilevanti per ogni caso d'uso;
- suddividere ciascun parametro in **categorie** e **scelte** (classi di equivalenza);
- generare **test frame** validi, ciascuno rappresentante un insieme coerente di scelte;
- derivare da questi i **test case** eseguibili e i rispettivi **oracoli** (risultati attesi)

2 Use case summary

2.1 UC-1 — Analisi di una directory di repository (senza cloning)

Descrizione	L'utente esegue solo l'analisi su una directory locale contenente repository già disponibili, ottenendo i risultati in `io/output/`.
Attori	Utente (ricercatore)
Entry condition	`main_args.py` disponibile; dipendenze installate; directory `--repository-path` accessibile (per scenari "success").
Exit condition	Successo: risultati salvati in `io/output/`. Errore: messaggio d'errore e assenza di output significativo.
Flusso di eventi principale	1. Invocazione con `--repository-path` e `--analysis`. 2. Validazione path. 3. Scansione e analisi dei repository. 4. Salvataggio output in `io/output/`.

2.2 UC-2 — Cloning (e verifica cloning) da lista CSV

Descrizione	L'utente clona repository elencati in un CSV e verifica gli esiti, salvando i repository in `io/repos/` e producendo log/report.
Attori	Utente (ricercatore)
Entry condition	`main_args.py` disponibile; dipendenze installate; CSV disponibile (per success); accesso a rete/Git; permessi scrittura in `io/repos/`.
Exit condition	Successo: repo clonati in `io/repos/` + report disponibili. Errore: messaggio d'errore e cloning non eseguito o parziale.
Flusso di eventi principale	1. Invocazione con `--project-list --clone --clone-check`. 2. Validazione CSV. 3. Clonazione repo in `io/repos/`. 4. Verifica cloning e report. 5. Terminazione.

3 UC-1 – Category Partition, Test Frame, Test Case

Parametri: Path della directory contenente i repository (`repository_path`)

Oggetti dell'ambiente: Filesystem, contenuto dei progetti appartenenti alla directory analizzata

Categorie e scelte:

Categoria	**ID**	**Descrizione**	**Proprietà / Vincoli**
Esistenza input directory	ID1	Directory non esiste	[Error]
	ID2	Directory esiste	[property DirOk]
Contenuto directory	CI0	Directory vuota (0 progetti)	[if DirOk] [property Empty]
	CI1	Directory con 1 progetto	[if DirOk] [property NonEmpty]
	CI2	Directory con più progetti (>1)	[if DirOk] [property NonEmpty, MultiProject]
Cardinalità Producer	NP0	0 producer	[if DirOk]
	NP1	1 producer	[if NonEmpty]
	NP2	2+ producer	[if MultiProject] [Single]
Cardinalità Consumer	NC0	0 consumer	[if DirOk]
	NC1	1 consumer	[if NonEmpty]
	NC2	2+ consumer	[if MultiProject] [Single]

3.1 Test Frame

ID	**Combinazioni (categorie/scelte)**	**Oracolo (risultato atteso)**
TF1	ID1	Visualizza messaggio di errore “Input folder not found”
TF2	ID2, CI0, NP0, NC0	Nessun progetto classificato
TF3	ID2, CI1, NP0, NC0	Progetto non classificato
TF4	ID2, CI1, NP1, NC0	Progetto classificato “Producer”
TF5	ID2, CI1, NP0, NC1	Progetto classificato “Consumer”
TF6	ID2, CI1, NP1, NC1	Progetto classificato “Producer + Consumer”
TF7	ID2, CI2, NP0, NC0	Nessun progetto classificato
TF8	ID2, CI2, NP1, NC0	Almeno un progetto classificato “Producer”
TF9	ID2, CI2, NP0, NC1	Almeno un progetto classificato “Consumer”
TF10	ID2, CI2, NP1, NC1	Almeno 1 “Producer” e 1 “Consumer” o 1 “Producer+Consumer”
TF11	ID2, CI2, NP2, NC1	2+ progetti classificati “Producer”
TF12	ID2, CI2, NP1, NC2	2+ progetti classificati “Consumer”

3.2 Test Case

TC	**Input (repository_path)**	**Descrizione ambiente**	**Oracolo**
TC1	`repo_does_not_exist`	Directory inesistente	Messaggio "Input folder not found"
TC2	`empty_repo`	Cartella vuota	Nessun progetto classificato
TC3	`test_repos`	1 progetto senza librerie ML	Nessun progetto classificato
TC4	`test_repos`	1 progetto con funzioni di training	Progetto classificato "Producer"
TC5	`test_repos`	1 progetto con funzioni di inferenza	Progetto classificato "Consumer"
TC6	`test_repos`	1 progetto con training e inferenza separati	Progetto classificato "Producer + Consumer"
TC7	`test_repos`	Più progetti, nessuno ML	Nessun progetto classificato
TC8	`test_repos`	Più progetti, almeno uno con training	Almeno un progetto "Producer"
TC9	`test_repos`	Più progetti, almeno uno con inferenza	Almeno un progetto "Consumer"
TC10	`test_repos`	Più progetti con training e inferenza	Almeno un "Producer" e un "Consumer"
TC11	`test_repos`	≥2 progetti con training	Almeno due "Producer"

TC12	`test_repos`	≥2 progetti con inferenza	Almeno due “Consumer”

4 UC-2 – Category Partition, Test Frame, Test Case

Parametri: Path del file CSV contenente l'elenco di repository (`project_list_path`)

Oggetti dell'ambiente: Filesystem, contenuto del file CSV specificato

Categorie e scelte:

Categoria	**ID**	**Descrizione**	**Vincoli**
Esistenza file CSV	EF1	File CSV non esiste	[Error]
	EF2	File CSV esiste	[property file_exists]
Contenuto CSV	CC1	CSV vuoto (solo header)	[if file_exists]
	CC2	CSV con 1 repository	[if file_exists]
	CC3	CSV con N repository (>1)	[if file_exists]

4.1 Test Frame

ID	**Combinazioni**	**Oracolo**
TF1	EF1	Nessun repository clonato (errore: file inesistente)
TF2	EF2, CC1	Nessun repository clonato (CSV vuoto)
TF3	EF2, CC2	1 repository clonato correttamente
TF4	EF2, CC3	N repository clonati correttamente

4.2 Test Case

TC	**Input (project_list_path)**	**Descrizione ambiente**	**Oracolo**
TC1	`non-existent.csv`	File CSV inesistente	Messaggio "Project list file not found"
TC2	`empty_projects.csv`	File CSV vuoto	Nessun progetto clonato
TC3	`single_project.csv`	Contiene `10up/classifai`	Il progetto *classifai* viene clonato correttamente
TC4	*multi_projects.csv*	Contiene `10up/classifai`, `2knal/Honesty`, `5hirish/adam_qas`	Tutti i repository vengono clonati correttamente

5 Conclusioni

Il testing di sistema effettuato secondo la tecnica **Category Partition** ha permesso di esplorare in modo sistematico tutte le combinazioni significative dei parametri di input e dello stato degli oggetti dell'ambiente, garantendo la copertura funzionale delle due principali funzionalità del tool:

1. **Analisi e classificazione** dei progetti ML;
2. **Clonazione** dei repository da file CSV.

Complessivamente sono stati definiti, implementati ed eseguiti **16 test case** (**12** per UC-1 *Analysis* e **4** per UC-2 *Cloning*), e l'esecuzione è terminata **senza alcun fallimento** (tutti i test case hanno soddisfatto gli oracoli attesi).

Tale base di test sarà utilizzata come riferimento per la successiva fase di **post-modifications system testing**, per verificare che le modifiche introdotte non compromettano il comportamento corretto del sistema.