Perfective and Corrective Reengineering Report for Mark

a.prisco50@studenti.unisa.it 0522501976
c.ranieri7@studenti.unisa.it 0522501977

June 30, 2025

1 Version History

Versione	Data	Descrizione
0.1	25/06/2025	Stesura della struttura del documento.
1.0	25/06/2025	Prima stesura del documento.

2 Riferimenti

All'interno del documento vengono citate le seguenti informazioni.

Riferimento	Descrizione
IGES_Mark_CCR	Documento relativo alla comprensione del codice in cui sono stati identificati una serie di problemi relativi ai dettagli tecnici del sistema.
IGES_Mark_CRD	Documento relativo alle change requests.

3 Introduzione

Di seguito sono riportate tutte le informazioni relative all'analisi, alla progettazione e alla realizzazione delle modifiche necessarie alla risoluzione della CR_1.

Obiettivo del Documento All'interno di questo documento saranno trattate le tecniche e i risultati delle attività di impact analysis al fine di comprendere le relazioni tra i moduli e come le modifiche definite nel documento **IGES_Mark_CCR** che devono essere applicate possono impattare su questi ultimi oltre che a una serie di informazioni atte a tenere traccia delle modifiche effettuate.

Contesto del Progetto Mark è un tool scritto in Python che dato un progetto, tramite l'ausilio di una Knowledge Base e delle detection rules, può classificare progetti ML in tre diverse categorie: ML-Consumer, ML-Producer e ML-Producer e Consumer.

4 Analisi dell'Impatto delle Modifiche

Impatto della CR_1 La CR_1 si basa sulla reingegnerizzazione del tool rendendolo Object-Oriented. La previsione è che la modifica coinvolgerà tutti i moduli e le funzioni che li compongono, le quali verranno trasformate in metodi, modificate o eliminate. Per effettuare questa analisi, abbiamo utilizzato la tecnica di **static impact analysis** dal punto di vista delle chiamate a funzioni e dei dati. In particolare, sono stati analizzati manualmente il codice sorgente di ogni modulo del sistema, permettendo di determinare quanto segue. Per trarre queste informazioni sono state utilizzate tutte le informazioni a disposizione nel documento **IGES_Mark_CCR**.

Moduli indipendenti dal punto di vista delle chiamate a funzione tra i moduli sono:

- *cloner.py*
- *cloning_check.py*
- *notebook_converter.py*
- *Results_Analysis.py*
- *merge.py*

Moduli con dipendenze:

- Il modulo *exec_analysis.py* crea un'istanza delle classi *MLConsumerAnalyzer.* e *MLProducer-Analyzer.*, sulle quali richiama rispettivamente i metodi *analyze_projects_set_for_consumers* e *analyze_projects_set_for_producers*.
- I moduli *MLConsumerAnalyzer.* e *MLProducerAnalyzer.* dipendono dal modulo *library_extractor.py* utilizzando rispettivamente le funzioni *get_libraries* e *check_ml_library_usage*.

Analisi delle dipendenze nell'utilizzo dei dati Dall'analisi dei dati, è stato possibile determinare che:

- Il modulo *cloner.py* prende in input un file con le repository da clonare e produce una copia delle repository in una directory.
- Il modulo *cloning_check.py* prende in input la directory delle repository clonate e un file con le repository da controllare e produce un report sulle repository clonate e quelle non clonate.
- Il modulo *exec_analysis.py* prende in input la directory delle repository clonate e produce una directory contenente i risultati dell'analisi per ogni repository clonata.
- Il modulo *Results_Analysis.py* prende in input i file di analisi delle repository e un file oracolo e produce un file che è un merge dei file di input.
- Il modulo *merge.py* prende in input i file di analisi delle repository e un file oracolo e produce dei file contenenti falsi positivi e negativi per i producer e consumer, e calcola varie metriche di analisi.

Nota. I vari moduli sono quindi temporalmente dipendenti l'uno dall'altro seguendo una serie di dipendenze a catena che coincidono con la pipeline già descritta nel documento **IGES_Mark_CCR**.

Analisi Dinamica Dopo la fase di static impact analysis, per confermare le informazioni estratte è stato effettuato anche un processo di **dynamic impact analysis**, data la ridotta dimensione del sistema. Il processo consiste nell'eseguire i moduli del sistema tramite il tool **Coverage.py**, che permette di determinare quali moduli e codice sono coinvolti durante l'esecuzione su dati reali. I dati utilizzati sono 341 repository, sulle quali si dispone di tutte le informazioni per eseguire la totalità dei moduli, dalla clonazione alla verifica della correttezza dell'analisi.

Questa analisi ha confermato i risultati della static analysis:

- Per l'esecuzione dei moduli *cloner.py*, *cloning_check.py*, *notebook_converter.py*, *Results_Analysis.py* e *merge.py*, l'unico codice coinvolto è quello dei moduli stessi.
- L'esecuzione del modulo *exec_analysis.py* ha coinvolto anche i moduli *consumer_classifier_by_dict.py*, *producer_classifier_by_dict.py* e *library_extractor.py*.

Conclusioni sull'impatto della CR_1 Sulla base dei risultati delle analisi statica e dinamica, la previsione iniziale risulta corretta: tutti i moduli e i relativi metodi/funzioni saranno coinvolti dalla modifica, data la natura a pipeline del sistema e la modifica da implementare.

Impatto della CR_2 La CR_2 consiste nell'aggiunta di una GUI che permette di utilizzare le funzionalità di analisi e clonazione offerte dal sistema esistente. L'analisi dell'impatto di questa modifica è stata effettuata sulla base dei risultati ottenuti dall'analisi statica e dinamica per la CR_1, e sulla base del funzionamento della GUI da implementare, il quale si basa su due passi principali:

1. Eseguire i moduli *cloner.py* e *exec_analysis.py*, dando in input la directory in cui clonare le repository, il file contenente le repository da clonare (la stessa su cui verrà effettuata l'analisi) e la directory in cui inserire i risultati dell'analisi.
2. Successivamente, al termine del processo di esecuzione dell'analisi, la GUI aprirà delle tab in cui visualizzare i risultati scritti all'interno dei file creati dal modulo di analisi.

Proprio per il modo in cui è strutturata l'interfaccia, che utilizza i risultati dei moduli esistenti così come sono, l'aggiunta della GUI non dovrebbe coinvolgere alcun modulo del sistema esistente.

5 Progettazione delle Modifiche

Le modifiche descritte nel documento **IGES_Mark_CRD** saranno applicate seguendo l'ordine del pipeline diagram già descritto nel documento **IGES_Mark_CCR**.

6 Reengineering del Modulo *cloner.py*

Durante l'attività di reengineering, il modulo *cloner.py* è stato completamente ristrutturato per adottare il paradigma orientato agli oggetti (OOP) diventando un classe. L'intervento si è reso necessario per migliorare l'organizzazione del codice, facilitarne l'estensione e garantire una migliore manutenibilità nel lungo termine.

Nota. Le variabili di istanza sono state identificate analizzando i parametri maggiormente ricorrenti tra le funzioni, con l'obiettivo di centralizzarne la gestione all'interno della classe.

Variabili di Istanza Per semplificare l'accesso ai dati fondamentali condivisi tra i metodi, sono state introdotte due variabili di istanza:

- *input_file*: rappresenta il path del file .csv contenente la lista delle repository GitHub da clonare;
- *output_path*: indica la directory nella quale verranno salvate le repository clonate.

Conversione di Funzioni in Metodi Tutte le principali funzioni del modulo sono state convertite in metodi di una nuova classe. Questo ha permesso di eliminare il passaggio esplicito di parametri ridondanti e aumentare la coesione del codice. Le conversioni effettuate sono le seguenti:

- *main(input_file, output_path)* $\rightarrow$ *run(self)*;
- *start_search(iterable, output_path, max_workers=None)* $\rightarrow$ *start_search(self, iterable, max_workers=None)*;
- *__search(row, lock, output_path)* $\rightarrow$ *__search(self, row, lock)*;
- *delete_repos(to_delete)* $\rightarrow$ funzione rimossa in quanto non utilizzata all'interno del flusso del programma.

Modifiche alla Funzione *main* La funzione *main*, in precedenza punto di ingresso del modulo, è stata sostituita dal metodo *run* della classe. Durante questa trasformazione sono state apportate le seguenti modifiche:

- Eliminazione delle variabili *already_analyzed* ed *error*, che risultavano inutilizzate;
- Aggiunta di un controllo preventivo sull'esistenza e correttezza del file .csv di input, al fine di migliorare la robustezza e prevenire comportamenti anomali in caso di path errato.

Conclusioni Il refactoring del modulo *cloner.py* ha reso il codice più modulare e facilmente estendibile. L'adozione dell'OOP ha favorito una migliore organizzazione delle responsabilità, ponendo le basi per un'integrazione più agevole con gli altri componenti del sistema.

7 Reengineering del Modulo *cloning_check.py*

Il modulo *cloning_check.py* è stato ristrutturato secondo il paradigma orientato agli oggetti per migliorarne la manutenzione e la chiarezza. L'intervento ha previsto la trasformazione delle funzioni procedurali in metodi di una classe e l'introduzione di variabili di istanza per centralizzare i dati condivisi.

Nota. Le variabili di istanza sono state selezionate analizzando i parametri più frequentemente utilizzati all'interno del modulo, così da semplificare la gestione dello stato interno.

Variabili di Istanza Sono state definite le seguenti variabili di istanza fondamentali:

- *input_file*: percorso del file contenente la lista delle repository da controllare per verificarne la clonazione;
- *output_path*: percorso della directory di output che contiene le repository clonate.

Conversione di Funzioni in Metodi Le funzioni originarie sono state convertite in metodi della nuova classe, eliminando così la necessità di passare ripetutamente i medesimi parametri:

- *main(input_file, input_path)* $\rightarrow$ *run(self)*;
- *clean_log()* $\rightarrow$ *clean_log(self)*;
- *get_effective_repos(path)* $\rightarrow$ *get_effective_repos(self)*;
- *count_effective_repos(path)* $\rightarrow$ *count_effective_repos(self)*;
- *get_cloned_list(df, path)* $\rightarrow$ *get_cloned_list(self, df)*;
- *get_not_cloned_list(df, path)* $\rightarrow$ *get_not_cloned_list(self, df)*;
- *check_cloned_repo(path, project_name)* $\rightarrow$ *check_cloned_repo(self, project_name)*.

Modifiche al Metodo *run* Nel metodo *run* è stato aggiunto un controllo preventivo sull'esistenza dei percorsi specificati nei parametri prima di procedere con le operazioni di verifica. Questo miglioramento ha aumentato la robustezza del sistema, evitando errori legati a path non validi o inesistenti.

Conclusioni La ristrutturazione del modulo ha permesso di organizzare il codice in maniera più modulare e chiara, facilitandone la manutenzione futura e l'integrazione con altri moduli del progetto.

8 Reengineering del Modulo *notebook_converter.py*

Il modulo *notebook_converter.py* è stato riorganizzato secondo il paradigma orientato agli oggetti, con l'obiettivo di migliorare la gestione del codice e facilitare l'estensione futura delle funzionalità.

Nota. Le variabili di istanza sono state scelte analizzando i parametri ricorrenti e quelli che definiscono lo stato necessario all'interno della classe.

Variabili di Istanza È stata definita la seguente variabile di istanza:

- *folder_path*: percorso della directory contenente le repository da cui convertire i notebook.

Conversione di Funzioni in Metodi Le funzioni procedurali sono state convertite in metodi della nuova classe, eliminando inoltre quelle non utilizzate:

- *convert_notebook_to_code(file)* $\rightarrow$ *convert_notebook_to_code(self, file)*;
- *convert_all_notebooks(folder_path)* $\rightarrow$ *convert_all_notebooks(self)*;
- *convert_and_check_notebook(file)*: funzione rimossa poiché non utilizzata.

Modifiche al Metodo *run* Nel metodo *run* è stato aggiunto un controllo sull'esistenza del percorso specificato da *folder_path* prima di avviare la conversione dei file, così da prevenire errori dovuti a path non validi o assenti.

Conclusioni Questa ristrutturazione ha migliorato la robustezza e la manutenibilità del modulo, rendendo più semplice l'integrazione con altri componenti e l'eventuale estensione futura.

9 Reengineering del Modulo *exec_analysis.py*

Il modulo *exec_analysis.py* è stato ristrutturato per adottare il paradigma orientato agli oggetti, migliorando la chiarezza e la gestione del flusso di esecuzione.

Nota. Le variabili di istanza sono state identificate considerando i parametri fondamentali per la configurazione e l'esecuzione del modulo.

Variabili di Istanza Sono state introdotte le seguenti variabili di istanza:

- *input_path*: percorso della directory contenente le repository da analizzare;
- *output_path*: percorso della directory dove saranno salvati i risultati dell'analisi.

Conversione di Funzioni in Metodi La funzione principale è stata convertita in un metodo di classe:

- *exec_analysis(input_path, output_path)* $\rightarrow$ *run(self)*.

Modifiche Implementate

- Le variabili *output_folders* e *producer_dict_paths*, originariamente array contenenti un singolo elemento, sono state semplificate al solo elemento contenuto per una gestione più diretta;
- Prima dell'esecuzione dell'analisi, è stato integrato il processo di esecuzione del modulo *NotebookConverter* per garantire la conversione necessaria dei file.

Conclusioni Queste modifiche hanno migliorato la robustezza del modulo e hanno reso più fluido il flusso di lavoro, garantendo che tutte le fasi preliminari vengano eseguite automaticamente prima dell'analisi vera e propria.

10 Reengineering delle Classi *MLProducerAnalyzer.*, *MLConsumerAnalyzer.* e *MLAnalyzerBase*

Per favorire il riutilizzo del codice e una maggiore manutenibilità, le classi *MLProducerAnalyzer.* e *MLConsumerAnalyzer.* sono state rifattorizzate estendendo una nuova classe astratta, *MLAnalyzerBase*, in cui sono stati centralizzati i metodi e le variabili comuni.

Variabili di Istanza

- La variabile *output_folder*, inizialmente definita nelle classi concrete, è stata spostata nella classe *MLAnalyzerBase*;
- È stata introdotta la variabile *analysis_type*, che assume i valori *producer* o *consumer*, per generalizzare i comportamenti che differivano solo in base a questa keyword.

Riorganizzazione dei Metodi Diversi metodi ridondanti presenti nelle classi concrete sono stati accorpati e definiti nella classe astratta:

- *init_producer_analysis_folder(self)* e *init_consumer_analysis_folder(self)* $\rightarrow$ *init_analysis_folder(self)*;
- *load_producer_library_dict(input_file)* e *load_consumer_library_dict(input_file)* $\rightarrow$ *load_library_dict(input_file: str)* (definito come @*staticmethod*);
- *build_regex_pattern(keyword)* e *baseline_check(project, dir, df)* sono stati spostati nella classe base e marcati come @*staticmethod*;
- Il metodo *check_for_training_method* è stato generalizzato in *check_training_method(self, file, library_dict_path)* e spostato nella classe base come metodo astratto.

Stato dei Metodi Nelle Classi Derivate

- **MLConsumerAnalyzer.** I metodi *init_consumer_analysis_folder*, *build_regex_pattern*, *baseline_check*, *load_consumer_library_dict*, e *check_training_method* sono stati rimossi in quanto sostituiti dalle versioni generalizzate;
- **MLConsumerAnalyzer.** I metodi *check_for_inference_method*, *analyze_single_file*, *analyze_project_for_consumers*, *analyze_projects_set_for_consumers* non hanno subito modifiche.
- **MLProducerAnalyzer.** I metodi *init_producer_analysis_folder*, *build_regex_pattern*, *baseline_check*, *load_producer_library_dict* e *check_for_training_method* sono stati sostituiti dalle relative implementazioni nella classe astratta;
- **MLProducerAnalyzer.** I metodi *analyze_single_file*, *analyze_project_for_producers*, *analyze_projects_set_for_producers* non hanno subito modifiche.

Classe Astratta *MLAnalyzerBase* La classe *MLAnalyzerBase* definisce i seguenti metodi condivisi:

- *init_analysis_folder(self)*;
- *build_regex_pattern(keyword: str)* (@*staticmethod*);
- *baseline_check(project: str, dir: str, df: pd.DataFrame)* (@*staticmethod*);
- *load_library_dict(input_file: str)* (@*staticmethod*);
- *check_training_method(self, file: str, library_dict_path: str)* (metodo astratto da implementare nelle sottoclassi).

Conclusioni Questa ristrutturazione ha permesso di:

- Ridurre la duplicazione del codice tra le due classi *MLConsumerAnalyzer.* e *MLProducerAnalyzer.*;
- Migliorare la leggibilità e la modularità, centralizzando nella superclasse le funzionalità comuni;
- Facilitare eventuali estensioni future del sistema con nuovi tipi di analizzatori.

11 Reengineering del Modulo *Results_Analysis.py*

Il modulo *Results_Analysis.py* è stato reingegnerizzato per eliminare la ripetizione di codice relativa alla gestione separata delle versioni *consumer* e *producer*. L'intervento si è concentrato sulla generalizzazione delle funzionalità comuni e sulla parametrizzazione degli aspetti variabili tramite apposite variabili di istanza.

Eliminazione dei Blocchi Ridondanti Tutti i blocchi di codice duplicati per la gestione separata delle analisi dei *producer* e dei *consumer* sono stati unificati. Questo ha consentito di ridurre la complessità e migliorare la leggibilità del codice.

Conversione di Funzioni in Metodi

- Introdotto il metodo *set_column_name_and_is_new_and_version(self, column_name, is_new, version)* che permette di configurare dinamicamente l'analisi, evitando duplicazioni e rendendo il modulo più flessibile.

Modifiche al Metodo *run* Tutta la logica dello script, il quale non era strutturato in funzioni, è stata centralizzata nel metodo *run*.

Benefici del Refactoring

- Riduzione della ridondanza attraverso la centralizzazione del comportamento comune;
- Migliore configurabilità del modulo, grazie alla separazione tra logica e parametri operativi;
- Maggiore robustezza e semplicità nel mantenere o estendere il codice in futuro.

12 Reengineering del Modulo *merge.py*

Nel processo di reingegnerizzazione del modulo *merge.py* sono stati rimossi elementi superflui e semplificata la logica del flusso, al fine di aderire più efficacemente al paradigma orientato agli oggetti.

Variabili di Istanza Le uniche variabili di istanza mantenute nella nuova implementazione sono:

- *column_name*: utilizzata per identificare il tipo di analisi condotta (*producer* o *consumer*);
- *result_name*: rappresenta il nome da assegnare ai file di output prodotti.

Variabili Eliminate Sono state rimosse le seguenti variabili, risultate non più necessarie alla luce della nuova struttura:

- *base_output_path*: eliminata poiché il path di output viene ora ricavato direttamente a partire dal contenuto della colonna;
- *analysis_path*: ritenuta ridondante e rimossa;
- *oracle_name*: non utilizzata all'interno del nuovo flusso.

Conversione di Funzioni in Metodi Le funzioni dedicate alla valutazione delle predizioni sono state convertite in metodi della nuova classe, con eliminazione dei parametri ridondanti come *column_name*:

- *get_false_positives(df, column_name)* → *get_false_positives(self, df)*;
- *get_false_negatives(df, column_name)* → *get_false_negatives(self, df)*;
- *calc_true_positives(df, column_name)* → *calc_true_positives(self, df)*;
- *calc_false_positives(df, column_name)* → *calc_false_positives(self, df)*;
- *calc_true_negatives(df, column_name)* → *calc_true_negatives(self, df)*;
- *calc_false_negatives(df, column_name)* → *calc_false_negatives(self, df)*;
- *calc_performance_metrics(df, column_name)* → *calc_performance_metrics(self, df)*;

Inoltre, sono state apportate le seguenti modifiche:

- La funzione *join(column_name, df_oracle, df_produced)* è stata eliminata in quanto non più necessaria nella nuova struttura a oggetti;
- La funzione *reporting(oracle_name, column_name, base_output_path, analysis_path)* è stata convertita nel metodo principale *run(self)* della classe.

Modifiche al Flusso del Modulo

- Il metodo *reporting*, precedentemente utilizzato per la scrittura dei risultati, è stato eliminato. La sua logica interna richiamava una funzione *join* che, a sua volta, produceva un file di output identico a quello già creato da *results_analysis.py*, rendendo l'operazione ridondante.
- È stato introdotto il metodo *set_column_name_and_version(self, column_name, version)*, che permette di configurare dinamicamente il tipo di analisi (*producer* o *consumer*) e la versione del file dei risultati da produrre.

13 Reengineering del Modulo *library_extractor.py*

Tra le modifiche apportate al modulo *library_extractor.py* sono da annoverare la definizione di una nuova variabile di istanza oltre alla conversioni delle funzioni esisteni in metodi di una nuova classe.

Variabili di Istanza La nuova implementazione mantiene un'unica variabile di istanza, utilizzata per l'analisi dei file:

- *file*: rappresenta il percorso del file da analizzare.

Conversione di Funzioni in Metodi Le funzioni originarie sono state convertite in metodi di istanza, eliminando la necessità di passare il parametro *file* e rendendo il codice più aderente al paradigma orientato agli oggetti:

- *get_libraries(file)* $\rightarrow$ *get_libraries(self)*;
- *check_ml_library_usage(file, library_dict)* $\rightarrow$ *check_ml_library_usage(self, library_dict, is_consumer=False)*.

Modifiche ai Parametri Nel metodo *check_ml_library_usage*, è stato aggiunto il parametro opzionale *is_consumer* (booleano), che consente di generalizzare la logica del metodo per essere compatibile sia con l'analisi dei consumer che dei producer. Questo approccio favorisce il riutilizzo del codice all'interno delle classi *MLConsumerAnalyzer* e *MLProducerAnalyzer*.