PROGETTO “JUNIT-TO-JMH”

Pre - Modification System Testing

Nome	Matricola
Benito Pigna	NF22500110
Michele Cillo	NF22500124

Versione: 1.0

Data: 20/02/2026

Sommario

1 INTRODUZIONE

l presente documento descrive l'attività di testing svolta sulla versione iniziale del sistema junit-to-jmh, precedente all'applicazione delle Change Request. L'obiettivo è verificare il corretto funzionamento dei moduli principali, dal parsing dell'AST alla generazione dei file JMH, e stabilire una baseline utile per i futuri test di regressione che verranno condotti dopo l'introduzione di nuove funzionalità. La parte di generazione di metriche avanzate o supporto a JUnit 5 è fuori dallo scope del presente system test: ci concentriamo unicamente sul tool originale (supporto esclusivo a JUnit 4). Oltre ai test di unità e di integrazione già effettuati sui componenti core, l'attenzione è qui posta sui test di sistema. Tutte le esecuzioni funzionali sono state condotte tramite interfaccia CLI, strumento principale per l'automazione della conversione.

1.1 AMBIENTE DI TESTING

- **Sistema Operativo:** macOS / Linux / Windows
- **Java Development Kit (JDK):** 17+ (Compatibilità toolchain)
- **Build Tool:** Gradle 8.x
- **Dipendenze:** JavaParser, JUnit 4, JMH Core
- **Repository:** junit-to-jmh

1.2 TIPOLOGIA DI TESTING

L'attività di testing è stata condotta adottando un approccio di tipo black-box, concentrandosi esclusivamente sul comportamento osservabile del sistema e lasciando da parte gli aspetti interni del codice, già coperti dai test unitari e di integrazione. Per la progettazione dei test è stato utilizzato il Category Partition Method, che ha permesso di individuare in maniera sistematica le combinazioni più significative relative ai path di input, alla configurazione della CLI e alla struttura delle classi da analizzare. L'insieme dei test così costruito rappresenta una base solida per le successive attività di regression testing

2 TEST DI SISTEMA

In questo capitolo descriviamo la suite di test di sistema (cioè i test funzionali "end-to-end") predisposta per il tool junit-to-jmh. Prima dell'esecuzione dei test di sistema, sono già state condotte attività di verifica tramite unit test e integration test sui componenti principali del tool (motore AST, suite builder). La suite di test di sistema qui presentata ha quindi l'obiettivo di completare la validazione complessiva, verificandone il corretto funzionamento a livello sistemico

2.1 PARAMETRI DEL CATEGORY PARTITION

Il criterio adottato è il Category Partitioning: individuiamo i principali parametri che caratterizzano gli scenari di utilizzo in input da linea di comando (CLI) per junit-to-jmh e, per ciascuno, definiamo le categorie logiche.

2.1.1 PARAMETRI DI INPUT DIRECTORY (PATH)

Questi parametri descrivono la validità dei percorsi passati tramite CLI.

- **Source Path (SP):** Determina la validità della cartella contenente i file sorgente Java (.java).
- **Class Path (CP):** Determina la validità della cartella contenente il bytecode pre-compilato (.class), necessario per la risoluzione dei tipi.
- **Output Path (OP):** Determina la raggiungibilità della directory di destinazione per la generazione dei file JMH.

2.1.2 PARAMETRI TARGET DI CONVERSIONE

Questi parametri descrivono le modalità con cui vengono passate le classi da convertire.

- **Modalità Target (MT):** Definisce se le classi vengono passate direttamente come argomenti posizionali o tramite l'ausilio di un file testuale esterno.
- **Validità Target (VT):** Specifica se il nome della classe passato alla CLI corrisponde effettivamente a un file esistente nel source path.

2.1.3 PARAMETRO CONTENUTO CLASSE

- **Contenuto Test (CT):** Analizza strutturalmente il file target, verificando la presenza di validi metodi di test (annotati con @Test di JUnit 4).

2.2 SCELTE E COMBINAZIONI

L'implementazione efficace del meccanismo di conversione richiede l'integrazione e la valutazione congiunta dei parametri definiti nella fase preliminare.

2.2.1 Parametri di Input Directory

CATEGORIA	SCELTE
Source Path (SP)	1. Percorso valido e leggibile 2. Percorso inesistente o sintatticamente errato [ERRORE]
Class Path (CP)	1. Percorso valido contenente bytecode 2. Percorso inesistente [ERRORE]
Output Path (OP)	1. Directory di destinazione accessibile 2. Directory inaccessibile o read-only [ERRORE]

2.2.2 Parametri di Target di Conversione

CATEGORIA	SCELTE
Modalità Target (MT)	1. Nomi classi via argomenti posizionali [proprietà CLI_ARGS] 2. File di testo tramite --class-names-file [proprietà TXT_FILE]
Validità Target (VT)	1. Nessun target specificato [ERRORE] 2. Almeno una classe validamente esistente 3. Classe inesistente / Errore di battitura [ERRORE]

2.2.3 Parametri di Contenuto Classe

CATEGORIA	SCELTE
Contenuto Test (CT)	1. Contiene annotazioni JUnit 4 valide [se VT2] 2. Classe vuota o senza test JUnit 4 [se VT2] 0. Non applicabile (Alias0) [se VT1 o VT3]

2.3. Test Frame

Il criterio di copertura è applicato in modo da generare test funzionalmente coerenti e il più possibile rappresentativi. I casi di test, completi dei relativi oracoli, sono stati derivati a partire dalle categorie individuate, selezionando valori ammissibili ed esplicitando, tramite la notazione 0 (Alias0), le scelte non applicabili (ad esempio, l'omissione dell'analisi del contenuto della classe CT0 quando la classe in input non esiste VT3). Le combinazioni sono state costruite imponendo il vincolo di una sola sorgente di errore per test frame, così da evitare scenari con errori multipli difficili da interpretare.

Test Case ID	Test Frame	Oracolo atteso
TC_01	**SP1, CP1, OP1, MT1, VT2, CT1**	L'utente passa parametri validi via riga di comando. Il tool completa l'analisi, converte l'intera classe JUnit 4 e genera il file JMH corrispondente.
TC_02	**SP2, CP1, OP1, MT1, VT2, CT1**	L'utente fornisce un source path inesistente. Il sistema segnala un errore sui percorsi di input e si interrompe senza operare.
TC_03	**SP1, CP2, OP1, MT1, VT2, CT1**	L'utente fornisce un class path inesistente. Il parser AST non riesce a risolvere le dipendenze e l'analisi fallisce.
TC_04	**SP1, CP1, OP2, MT1, VT2, CT1**	L'utente fornisce un output path inaccessibile. Il sistema analizza il file ma non riesce a salvare il benchmark generato.
TC_05	**SP1, CP1, OP1, MT1, VT1, CT0**	L'utente non fornisce alcun nome di classe da analizzare. L'esecuzione si interrompe istantaneamente per mancanza di argomenti obbligatori.
TC_06	**SP1, CP1, OP1, MT1, VT3, CT0**	L'utente richiede la conversione di una classe il cui nome non è presente nel sourcePath. Il sistema segnala l'assenza del file corrispondente all'AST.

TC_07	**SP1, CP1, OP1, MT2, VT2, CT1**	L'utente fornisce i path validi e passa la lista delle classi tramite il flag --class-names-file. Il tool itera sul file testuale e converte tutte le classi elencate in batch.
TC_08	**SP1, CP1, OP1, MT1, VT2, CT2**	L'utente passa una classe esistente che però non contiene annotazioni @Test (vuota). Il tool analizza l'AST ma non genera benchmark JMH validi (output vuoto o ignored).