PROGETTO “JUNIT-TO-JMH”

Master Test Plan Document

Nome	Matricola
Benito Pigna	NF22500110
Michele Cillo	NF22500124

Versione: 1.0

Data: 27/02/2026

Sommario

1 INTRODUZIONE

ju2jmh è un framework Java progettato per la conversione automatica di classi di test JUnit in classi di benchmark JMH, facilitando il processo di creazione di test suite per le prestazioni (Microbenchmark). Il funzionamento originale si articola nell'identificazione dei metodi JUnit e nella generazione di una superclasse esecutiva. Il presente documento (Master Test Plan) definisce la strategia complessiva e le attività pianificate per la validazione delle Change Request (CR) applicate al sistema per risolverne le limitazioni storiche (mancato supporto a JUnit 5, assenza di granularità sui metodi e sovrascrittura distruttiva dei file). L'obiettivo è garantire che le nuove funzionalità (CR01, CR02, CR03) siano implementate correttamente senza introdurre anomalie nel comportamento preesistente del tool.

2 STRATEGIE DI TESTING

Il piano di testing adotta un approccio ibrido, combinando verifiche strutturali e funzionali a diversi livelli di astrazione.

2.1 TESTING DI UNITA' E INTEGRAZIONE

Per la verifica dei singoli componenti interni (come il parser dell'AST, il NestedBenchmarkSuiteBuilder e la WrapperBenchmarkFactory) viene impiegato un approccio White-Box. Viene mantenuta e aggiornata la suite dei test unitari esistenti. L'obiettivo è validare la corretta manipolazione dei nodi JavaParser (MethodDeclaration, TypeDeclaration) e l'interazione tra i moduli di generazione. Per isolare le dipendenze durante i test d'unità viene utilizzato il framework Mockito.

2.2 TESTING DI SISTEMA

Per la validazione end-to-end del convertitore verrà impiegato un approccio Black-Box. Questa strategia ignora la struttura interna del codice per focalizzarsi esclusivamente sulla correttezza dell'input fornito tramite Command Line Interface (CLI) e dell'output generato su disco. La progettazione dei casi di test di sistema seguirà il Category Partition Method, individuando in modo sistematico le combinazioni più rilevanti dei parametri di input (validità dei path, sintassi dei metodi richiesti, flag di risoluzione conflitti). Per validare l'output verrà impiegato un Golden Set: un insieme di file .java pre-calcolati e validati manualmente che fungerà da "Oracolo". Il test avrà successo se e solo se l'output generato dal tool sarà sintatticamente e semanticamente equivalente al file presente nel Golden Set.

2.3 TESTING DI REGRESSIONE

Data la natura architetturale delle modifiche previste e il forte accoppiamento del flusso di generazione, verrà adottato l'approccio **Retest-All** per garantire l'assenza di regressioni. Questa tecnica garantisce la *safety* assoluta: ad ogni modifica significativa (commit), verrà rieseguita l'intera suite lanciando il comando Gradle :test (che include unit e system test) per rilevare immediatamente eventuali anomalie.

3 STRUMENTI IMPIEGATI

L'infrastruttura di testing si avvale dei seguenti strumenti standard di mercato:

- **JUnit (4/5):** Runner principale per l'esecuzione della suite di test.
- **JaCoCo:** Strumento per l'analisi statica e la misurazione della Code Coverage.
- **PITest:** Strumento per l'esecuzione del Mutation Testing, utilizzato per valutare la robustezza e l'efficacia dei casi di test.
- **Mockito:** Framework per l'isolamento delle dipendenze (Mocking) nei test unitari.

4 COVERAGE E BASELINE

Prima dell'implementazione delle modifiche, è stata stabilita una **Baseline** analizzando la Test Suite esistente tramite l'impiego di strumenti di verifica dinamica:

- **Code Coverage (JaCoCo):** La copertura globale del sistema si attesta al 77%. Il package core (se.chalmers.ju2jmh) presenta una copertura solida dell'80%, mentre il package model risulta più debole (48%).
- **Robustezza (PITest):** È stata misurata la mutazione del codice per confermare l'efficacia dei test originali nel rilevare difetti strutturali.

L'obiettivo per la fase di Post-Modification Testing è mantenere la copertura del package *core* ≥80%, garantendo che tutto il nuovo codice introdotto per la gestione di JUnit 5, per il supporto Amber (Java 17) e per la logica di validazione/merge venga adeguatamente coperto dai nuovi test.

5 CRITERI DI ACCETTAZIONE

Il processo di testing si riterrà concluso con successo al soddisfacimento dei seguenti criteri:

1. **Regressione:** Superamento del 100% dei test della suite originale (verifica su input JUnit 4 classici).
2. **Validazione Golden Set:** I benchmark generati per i nuovi flussi (JUnit 5 e Wrapper mode) combaciano con gli oracoli del Golden Set.
3. **Fail-Fast:** Il sistema blocca correttamente e in modo sicuro input errati (es. metodi non presenti nell'AST) senza alterare il file system.
4. **I/O Sicuro:** Le operazioni di Merge e Overwrite creano correttamente i file di backup (.bak) prevenendo la perdita di dati.