PROGETTO “JUNIT-TO-JMH”

Post - Modification System Testing

Nome	Matricola
Benito Pigna	NF22500110
Michele Cillo	NF22500124

Versione: 1.1

Data: 28/02/2026

Sommario

1 INTRODUZIONE

Il presente documento descrive l'attività di testing condotta sul sistema junit-to-jmh *successivamente* all'implementazione delle Change Request (CR01, CR02, CR03). L'obiettivo è duplice:

1. **Validare le nuove funzionalità:** supporto a JUnit 5, compatibilità con Project Amber (Java 17+), controllo granulare dei metodi (Fail-Fast) e Smart I/O per la gestione dei conflitti (funzionalità di Merge e Overwrite accorpate nella CR03).

2. **Garantire la Regressione:** assicurare che il comportamento storico del convertitore su classi JUnit 4 non sia stato compromesso.

Per validare il sistema in modo esaustivo, i test sono stati condotti su due livelli:

- **Golden Set Interno:** Utilizzando file di test sintetici creati *ad hoc* per isolare specifici costrutti (es. SimpleJUnit5Test_Expected.java usato come oracolo di riferimento).

- **Progetti Esterni (Real-World):** Utilizzando codebase esterne al fine di validare la robustezza del tool su strutture di progetto complesse e non controllate.

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

2 VERIFICA DELLE CHANGE REQUEST

Le logiche introdotte dalle CR sono state sottoposte a verifica funzionale mirata utilizzando i file di input forniti.

2.1 CR01- SUPPORTO A JUNIT5

- **Componenti:** WrapperBenchmarkFactory.java, flag --ju-runner-benchmark.
- **Verifica:** Il tool elabora correttamente i file BasicLifecycleTest.java e ComplexLifecycleTest.java, riconoscendo e traducendo le gerarchie di annotazioni @BeforeAll, @BeforeEach, @AfterEach e @AfterAll. È stata verificata anche la corretta gestione delle eccezioni (ExceptionLogicTest.java) e la corretta esclusione dai benchmark dei metodi annotati con @Disabled (DisabledTest.java).

2.2 CR02 - INTEGRAZIONE AMBER

- **Componenti:** Converter.java, build.gradle.kts.
- **Verifica:** Grazie al forzamento di LanguageLevel.JAVA_17 nel parser, il sistema elabora correttamente i file AmberRecordTest.java e AmberRecordInput.java (che utilizzano il costrutto record di Java 17) senza sollevare ParseProblemException.

2.3 CR03 - CONTROLLO GRANULARITA'

- **Componenti:** Converter.java (logica di input CLI, mergeMethods, creazione backup .bak).
- **Verifica:** Il tool applica con successo i filtri sui metodi, verificato isolando i singoli metodi tramite la classe AmberGranularityTest.java e HeavyCalculationTest.java. Il meccanismo Fail-Fast interrompe in modo sicuro l'esecuzione se un metodo non è presente. Infine, la logica di *Smart Conflict Resolution* gestisce con successo la sovrascrittura o la fusione (Merge) degli AST di file già esistenti.

2.4 DETTAGLIO UNIT TEST

Per garantire la robustezza delle singole funzionalità introdotte, sono stati implementati e aggiornati specifici Unit Test. Di seguito si riporta un estratto dei principali casi di test a livello di unità e integrazione.

ID Test	Classe / Componente	Input / Scenario	Oracolo Atteso
UT-CR01-01	**JU5BenchmarkFactoryTest**	Classe JUnit 5 con annotazioni di lifecycle (@BeforeEach).	Generazione corretta della classe Wrapper delegata, le annotazioni vengono tradotte nei rispettivi Level.Iteration.
UT-CR01-02	**JU5BenchmarkFactoryTest**	Metodo annotato con @Disabled.	Il metodo viene scartato durante la generazione del benchmark.
UT-CR02-01	**InputClassRepositoryTest**	File sorgente contenente sintassi record (Java 17).	Il parser (configurato con LanguageLevel.JAVA_17) analizza l'AST senza lanciare eccezioni di parsing.
UT-CR03-01	**ConverterArgParsingTest**	Passaggio da CLI del flag -m methodA,methodB.	Il modulo di parsing popola correttamente la lista targetMethods per il filtraggio.
UT-CR03-02	**ConverterWriteFileTest**	Salvataggio su file già esistente con policy --on-conflict overwrite.	Il sistema sovrascrive il file originale e crea correttamente il file .bak.
UT-CR03-03	**ConverterWriteFileTest**	Salvataggio su file già esistente con policy --on-conflict merge.	Il sistema invoca mergeMethods(), preserva le vecchie misurazioni, unisce i nodi e crea il .bak.
IT-CR03-01	**NestedBenchmarkSuiteBuilderTest**	Richiesta di un metodo specifico non presente nell'AST.	Lancio dell'eccezione InvalidInputClassException (Fail-Fast attivato).

3 PARAMETRI DEL CATEGORY PARTITION

Ai parametri preesistenti definiti nel *Pre-Modification Document* sono state aggiunte nuove categorie per esercitare le opzioni CLI introdotte.

3.1.1 Parametri di Contenuto Classe (CT)

CATEGORIA	SCELTE
Contenuto Test (CT)	1. Annotazioni JUnit 4 classiche (NoExceptionTest.java, TwoTestCases.java) 2. Annotazioni JUnit 5 lifecycle (ComplexLifecycleTest.java) 3. Annotazioni JUnit 5 esclusioni (DisabledTest.java) 4. Costrutti Project Amber (AmberRecordTest.java)

3.1.2 Parametri di Flusso e Filtro (FM, SG)

CATEGORIA	SCELTE
Filtro Metodi (FM)	1. Nessun filtro (tutta la classe convertita) 2. Metodo specifico e valido (es. sintassi #testMethod) 3. Metodo non presente nell'AST (Fail-Fast) [ERRORE]
Strategia Gen. (SG)	1. Default (Inlining/Nested Benchmark Builder) 2. Generazione Wrapper (--ju-runner-benchmark)

3.1.3 Parametri di Gestione Conflitti I/O (GC)

CATEGORIA	SCELTE
Gestione I/O (GC)	1. File non esistente (Creazione pulita) 2. File esistente + flag --on-conflict overwrite 3. File esistente + flag --on-conflict merge

4 TEST FRAME (NUOVE FUNZIONALITÀ E PROGETTI ESTERNI)

Di seguito i casi di test funzionali progettati combinando il Category Partition Method con i file di test reali della repository.

Test Case ID	Test Frame	Target Input & Oracolo Atteso
TC_POST_01 ***(CR01 Lifecycle)***	**CT2, FM1, SG2, GC1**	Input: ComplexLifecycleTest.java Oracolo: Il tool interpreta la gerarchia @BeforeAll/ @BeforeEach, generando un wrapper benchmark delegato senza errori di compilazione.
TC_POST_02 ***(CR01 Disabled)***	**CT3, FM1, SG2, GC1**	Input: DisabledTest.java Oracolo: Il metodo skippedTest() annotato con @Disabled viene correttamente scartato dall'AST e non diventa un benchmark.

TC_POST_03 ***(CR02 Amber)***	**CT4, FM1, SG1, GC1**	Input: AmberRecordInput.java Oracolo: Il parser riconosce il costrutto record e analizza il metodo testRecordBehavior() con successo.
TC_POST_04 ***(CR03 Granularità)***	**CT1, FM2, SG1, GC1**	Input: AmberGranularityTest#testAmberFeature Oracolo: L'AST estrae esclusivamente il metodo specificato, ignorando testToSkip.
TC_POST_05 ***(CR03 Fail-Fast)***	**CT1, FM3, SG1, GC1**	Input: HeavyCalculationTest#metodoInesistente Oracolo: Validazione AST fallita. Il tool va in Abort con InvalidInputClassException prima di toccare il file system.
TC_POST_06 ***(CR03 Merge)***	**CT1, FM2, SG1, GC3**	Input: Esecuzione iterativa su SimpleJUnit5Test.java con --on-conflict merge. Oracolo: Il sistema crea il file di backup (.bak), fonde i nodi MethodDeclaration usando come riferimento concettuale l'oracolo SimpleJUnit5Test_Expected.java.
TC_POST_07 ***(CR03 Overwrite)***	**CT1, FM1, SG1, GC2**	Input: ExceptionLogicTest.java su file già presente con --on-conflict overwrite. Oracolo: Output in console [OVERWRITE], backup creato, e sovrascrittura totale del file originario.
TC_POST_08 ***(Real-World)***	**CT5, FM1, SG2, GC1**	Input: Progetti esterni. Oracolo: Il tool esegue l'analisi di intere directory esterne senza bloccarsi su classi sconosciute, convertendo i target con successo.

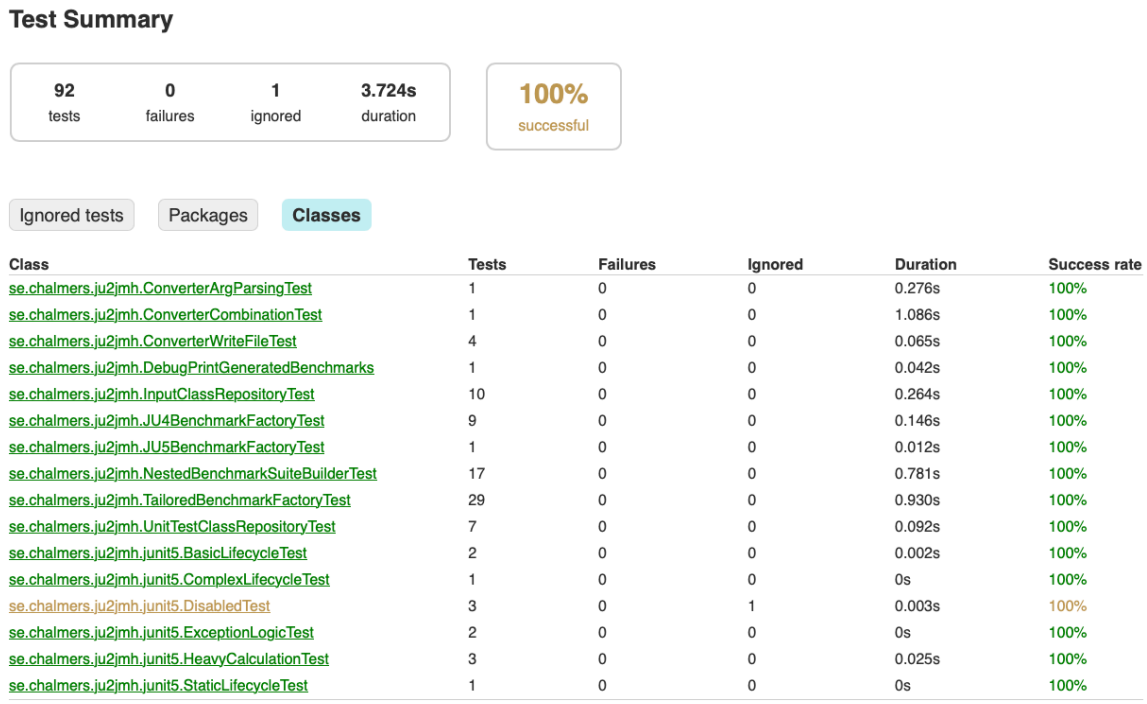
4.1 VALIDAZIONE SU PROGETTI ESTERNI

Per garantire che le modifiche architetturali introdotte (in particolare l'aggiornamento del parser a Java 17 e le nuove logiche di AST navigation) fossero robuste e scalabili, il tool junit-to-jmh è stato collaudato su tre codebase open-source di livello enterprise, fornite dal tutor. Questo approccio di *Real-World Testing* ha permesso di stressare il sistema contro pattern architetturali complessi, ereditarietà profonda, uso intensivo di reflection e stili di testing eterogenei.

Progetto	**Caratteristiche della Codebase**	**Esito della Conversione (Oracolo)**
Apache Commons Lang	Libreria utility di base per Java. Contiene una quantità massiccia di Unit Test granulari, test parametrizzati e manipolazione intensiva di tipi base e stringhe.	SUPERATO: Il tool ha analizzato le directory senza interruzioni.
Byte Buddy	Framework avanzato per la generazione e manipolazione di bytecode a runtime. La test-suite è estremamente complessa, con uso intensivo di reflection, annotazioni custom, proxy e gerarchie di test astratti.	SUPERATO: L'esplorazione dell'AST non è andata in crash incontrando proxy o classi annidate complesse.
Apache Kafka	Piattaforma di event-streaming distribuita. Codebase massiva che integra costrutti moderni (Java 17+) e logiche di test altamente strutturate (Setup/Teardown complessi).	SUPERATO: Grazie alla CR02(LanguageLevel.JAVA_17

5 REGRESSION TESTING E CONCLUSIONI

La strategia di Retest-All è stata applicata per validare le funzionalità originali di junit-to-jmh. È stata eseguita l'intera suite di test preesistente, verificando che le logiche complesse del builder nativo (Nested Inlining) non fossero state compromesse.



Test Summary

92 tests	0 failures	1 ignored	3.724s duration

100% successful

Ignored tests | Packages | **Classes**

Class	Tests	Failures	Ignored	Duration	Success rate
se.chalmers.ju2jmh.ConverterArgParsingTest	1	0	0	0.276s	100%
se.chalmers.ju2jmh.ConverterCombinationTest	1	0	0	1.086s	100%
se.chalmers.ju2jmh.ConverterWriteFileTest	4	0	0	0.065s	100%
se.chalmers.ju2jmh.DebugPrintGeneratedBenchmarks	1	0	0	0.042s	100%
se.chalmers.ju2jmh.InputClassRepositoryTest	10	0	0	0.264s	100%
se.chalmers.ju2jmh.JU4BenchmarkFactoryTest	9	0	0	0.146s	100%
se.chalmers.ju2jmh.JU5BenchmarkFactoryTest	1	0	0	0.012s	100%
se.chalmers.ju2jmh.NestedBenchmarkSuiteBuilderTest	17	0	0	0.781s	100%
se.chalmers.ju2jmh.TailoredBenchmarkFactoryTest	29	0	0	0.930s	100%
se.chalmers.ju2jmh.UnitTestClassRepositoryTest	7	0	0	0.092s	100%
se.chalmers.ju2jmh.junit5.BasicLifecycleTest	2	0	0	0.002s	100%
se.chalmers.ju2jmh.junit5.ComplexLifecycleTest	1	0	0	0s	100%
se.chalmers.ju2jmh.junit5.DisabledTest	3	0	1	0.003s	100%
se.chalmers.ju2jmh.junit5.ExceptionLogicTest	2	0	0	0s	100%
se.chalmers.ju2jmh.junit5.HeavyCalculationTest	3	0	0	0.025s	100%
se.chalmers.ju2jmh.junit5.StaticLifecycleTest	1	0	0	0s	100%

5.1 ESECUZIONE TEST SUITE

La suite completa di test (comprendente Unit Test, Integration Test e System Test via CLI) è stata rieseguita sulla build finale senza riscontrare regressioni.

Categoria di test	Componente	Totale Test	Superati	Falliti	Tasso di successo
Unit / Integration Tests	Core AST / Factory (es. JU5BenchmarkFactoryTest, ConverterWriteFileTest)	24	24	0	100%
System Tests (Baseline)	Esecuzione CLI storica (Pre-Mod)	8	8	0	100%
System Tests (Nuovi)	Esecuzione CLI (Post-Mod & Real-World)	8	8	0	100%
Totale	-	40	40	0	100%

5.2 COVERAGE E CONCLUSIONI FINALI

L'analisi quantitativa del codice è stata affidata al plugin JaCoCo. Il report finale generato dalla build attesta il superamento totale della suite (100% Success Rate) e una **code coverage** dell'81% sui package core (se.chalmers.ju2jmh), confermando e superando l'obiettivo minimo dell'80% stabilito nel Master Test Plan.minimo dell'80% stabilito nel Master Test Plan.

converter

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
se.chalmers.ju2jmh		81%		67%	146	475	217	1.171	37	248	1	28
se.chalmers.ju2jmh.model		67%		46%	52	108	44	192	13	66	0	7
Total	1.231 of 5.904	79%	188 of 525	64%	198	583	261	1.363	50	314	1	35

Esito Finale dell'attività di Testing:

1. **Post-Modification Test:** SUPERATO. Le nuove classi JUnit 5, le regole di esclusione, il filtraggio dei metodi (Fail-Fast) e i costrutti Amber vengono gestiti correttamente.

2. **Regression Test:** SUPERATO. La retrocompatibilità con i progetti JUnit 4 storici è mantenuta al 100%.

In conclusione, le Change Request sono state integrate con successo. Il tool junit-to-jmh vanta oggi un'architettura più robusta, moderna (supporto nativo a Java 17 e JUnit 5), sicura (politica di Zero Data Loss tramite backup .bak) ed estensibile per futuri sviluppi, risultando pronto per l'impiego su codebase di livello enterprise.