

Relazione Software Testing

Luca Falasca 0334722

Indice

1	Introduction	1
2	Bookkeeper	1
2.1	Journal	1
2.1.1	Descrizione	1
2.1.2	Category Partition	1
2.1.3	Boundary Analysis	2
2.1.4	Adequacy Control Flow	2
2.1.5	Adequacy Data Flow	3
2.1.6	Mutation Testing	3
2.1.7	Reliability	4
2.2	FileInfo	4
2.2.1	Descrizione	4
2.2.2	Category Partition	5
2.2.3	Boundary Analysis	5
2.2.4	Adequacy Control Flow	5
2.2.5	Adequacy Data Flow	6
2.2.6	Mutation Testing	6
2.2.7	Reliability	7
3	Avro	8
3.1	Schema	8
3.1.1	Descrizione	8
3.1.2	Category Partition	8
3.1.3	Boundary Analysis	8
3.1.4	Adequacy Control Flow	8
3.1.5	Adequacy Data Flow	9
3.1.6	Mutation Testing	9
3.1.7	Reliability	9
3.2	SchemaCompatibility	9
3.2.1	Descrizione	9
3.2.2	Category Partition	10
3.2.3	Boundary Analysis	10
3.2.4	Adequacy Control Flow	10
3.2.5	Adequacy Data Flow	10
3.2.6	Mutation Testing	11
3.2.7	Reliability	11
3.3	IT Schema-SchemaResolver	11

Elenco delle tabelle

1	Journal: Category Partition	12
2	Journal: Boundary Analysis	12
3	Journal: Test Suite - Category Partition	12
4	Journal: Test Suite - Adequacy Control Flow 1	13
5	Journal: Test Suite - Adequacy Control Flow 2	13

6	Journal: Test Suite - Adequacy Control Flow 3	13
7	Journal: Test Suite - Adequacy Control Flow 4	13
8	Journal: Test Suite - Adequacy Data Flow 1	13
9	Journal: Test Suite - Adequacy Data Flow 1	13
10	FileInfo: Boundary Analysis 1	13
11	FileInfo: Boundary Analysis 2	13
12	FileInfo: Test Suite - Category Partition	14
13	FileInfo: Test Suite - Adequacy Control Flow	14
14	FileInfo: Test Suite - Adequacy Data Flow	14
15	Schema - Category Partition	14
16	Schema - Category Partition	14
17	CreateRecord: Test Suite - Category partition	15
18	CreateRecord: Test Suite - Adequacy Control Flow	15
19	CreateRecord: Test Suite - Adequacy Control Flow	15
20	CreateRecord: Test Suite - Mutation Testing	15
21	SchemaCompatibility: Test Suite - Category Partition	15

Elenco delle figure

1	Diagrammi di listJournalIds	16
2	Jacoco coverage dilistJournalIds	16
3	Jacoco coverage dilistJournalIds	17
4	Jacoco coverage dilistJournalIds	17
5	Codice sorgente di JournalTest	17
6	Badua coverage dilistJournalIds	18
7	Badua coverage dilistJournalIds	18
8	Jacoco coverage dilistJournalIds	18
9	Pitest coverage dilistJournalIds	19
10	Mutazioni rilevate da Pitest	19
11	Mutazione inserita nel codice sorgente di Bookkeeper	20
12	Codice sorgente di JournalTest aggiornato con il mock	20
13	Jacoco coverage readHeader	20
14	Codice sorgente di FileInfoTest	21
15	Fallimento test readHeader	22
16	Jacoco coverage readHeader	23
17	Jacoco coverage readHeader	23
18	Badua coverage readHeader	23
19	Def-use not covered	24
20	Def-use covered	24
21	Badua coverage readHeader after test update	24
22	Pit coverage	24
23	Pit coverage	25
24	Pit coverage	25
25	Documentazione Record	26
26	Fallimento del 4 e 6 test	26
27	Jacoco coverage createRecord	26
28	Jacoco coverage Record Schema	26
29	Jacoco coverage Costruttore Record Schema e setField	26
30	Jacoco copertura setField	27
31	Jacoco coverage dopo incremento	27
32	Badua coverage createRecord	28
33	Badua coverage createRecord	29
34	PitCoverage	29
35	Mutazioni rilevate da Pitest	29
36	Mutazioni rilevate da Pitest	29
37	Mutazioni rilevate da Pitest	30
38	Mutazione in considerazione	31

39	Mutazione in considerazione coperta	31
40	Coverage Jacoco	31
41	Coverage Jacoco	31
42	Coverage Badua	31
43	Coverage Pit	32
44	Codice Integration Test	32
45	Report Integration Test	32

1 Introduction

Questa relazione descrive il lavoro di testing e di analisi di metriche di coverage effettuati sui due progetti open source BookKeeper e Avro. Per ogni progetto sono state prese in considerazione due classi, e per ognuna di esse si è fatto un lavoro di stesura di casi di test per poi andare a valutare l'adeguatezza tramite diverse metriche di coverage.

La struttura di ogni analisi è formata dai seguenti punti:

- Descrizione della classe e del metodo preso in considerazione
- Category Partition
- Boundary Analysis
- Adequacy Control Flow
- Adequacy Data Flow
- Mutation Testing
- Reliability

Per ognuna delle fasi (ad eccezioni delle prime due), se ritenuto necessario, sono state effettuate delle modifiche alla test suite basate sulle metriche di coverage calcolate, in modo da migliorare l'adeguatezza dei casi di test.

I framework utilizzati per calcolare la coverage sono Jacoco, Badua e Pitest. Essi sono stati implementati anche come CI sul repository github utilizzando le github actions.

I fork dei repository su cui è stato effettuato il lavoro di testing sono:

- [Bookkeeper](#)
- [Avro](#)

2 Bookkeeper

2.1 Journal

2.1.1 Descrizione

Path: org.apache.bookkeeper.bookie.Journal.java

Metodo considerato: listJournalIds

Documentazione a disposizione:

Lists all journal IDs filtered by a specified journal ID filter.

This method scans the given directory containing journal log files and extracts journal IDs based on the provided filter. If no filter is provided, all journal IDs present in the directory are returned.

Input:

- journalDir journal dir : The directory containing journal log files.
- filter journal id filter

Output: list of filtered ids

2.1.2 Category Partition

La category partition del metodo listJournalIds è descritta nella tabella 1.

Siccome la variabile journalDir è il path di una directory, ho partizionato il dominio in base al contenuto della directory e al suo effettivo utilizzo, rendendo la scelta delle partizioni una conseguenza del fatto che il metodo lavora su file di log.

2.1.3 Boundary Analysis

Andiamo a definire per ogni partizione i Boundary values. I valori scelti sono descritti nella tabella 2.

Per quanto riguarda il valore `JournalRollingFilter`, esso è l'unico filtro esistente utilizzato nell'applicazione, tuttavia siccome viene utilizzato in un contesto molto specifico dell'applicazione e non è un filtro generico, l'ho rimpiazzato con una sua versione semplificata più generale. Altrimenti non sarebbe stato adatto ad un test di unità, ma sarebbe stato più un test di integrazione.

Per quanto riguarda il valore `MyFilter`, esso è un filtro personalizzato creato appositamente per questo test che va a filtrare i journal in base al loro nome, in particolare se il `journalId` è >0 viene accettato, altrimenti no. Questo filtro ha lo scopo di testare come si comporta il sistema in caso di definizione di un nuovo filtro non già esistente nel sistema e quindi non è importante il tipo di filtraggio che fa.

Siccome i due parametri di input sono abbastanza scorrelati tra loro ha più senso adottare un approccio unidimensionale piuttosto che uno multidimensionale che sarebbe più adatto quando ci sono delle interazioni forti e che portano alla necessità di testare tutte le combinazioni tra i parametri. Inoltre avendo un approccio multidimensionale si finirebbe probabilmente per avere molti test non rilevanti e vanno a coprire scenari già coperti, e quindi sarebbero inutili.

Enumerano ora i casi di test derivanti da quest'analisi (Tabella 3)

Codice sorgente di `JournalTest` (Figura 5)

2.1.4 Adequacy Control Flow

Ora per verificare l'adeguatezza dei casi di test, vado a definire dei criteri di adeguatezza. Dalla documentazione sappiamo che questo metodo ricava gli id dei journal in una determinata cartella, filtrandoli in base ad un filtro specificato, se non viene specificato nessun filtro, allora vengono restituiti tutti gli id dei journal presenti nella cartella.

La Figura 1 mostra un diagramma funzionale del metodo `listJournalIds`. A partire da questo definisco i seguenti criteri di adeguatezza:

- almeno un test con un filtro
- almeno un test senza filtro
- almeno un test senza journal nella cartella
- almeno un test con almeno un journal nella cartella

In questo caso i criteri di adeguatezza sono già stati soddisfatti dai casi di test precedentemente descritti.

Dal diagramma precedente possiamo ricavare un control flow graph (Figura 1), per poi utilizzare dei criteri di copertura basati sul control flow.

Dato che in questa fase sto adottando un approccio black box e quindi non sto utilizzando il codice sorgente per verificare l'adeguatezza dei casi di test, ipotizzo in base agli input se un caso di test andrà a coprire un certo arco del CFG o meno. Per lo stesso motivo sto quindi basando questa parte dell'analisi esclusivamente sulla funzionalità del metodo, e dunque eviterò di utilizzare la coverage come parametro di adeguatezza, in quanto non è possibile ricavare dall'output del metodo se un certo arco è stato coperto o meno, ma solo se il risultato è corretto oppure no. Quindi valuterò la copertura del CFG solo in una fase successiva, quando prenderò in considerazione anche il codice sorgente (approccio whitebox), utilizzando la coverage come parametro di adeguatezza. Quello che farò in questa fase è quindi valutare se i casi di test che ho definito in precedenza coprono tutti gli archi del CFG, in base alla mia conoscenza del metodo e dei suoi input, e nel caso aggiungere quelli mancanti.

Arco (1,2) → Tutti i test che hanno almeno un file di log coprono questo caso, quindi non è necessario aggiungere ulteriori test.

Arco (2,3) → Tutti i test che hanno almeno un file di log e un filtro non null coprono questo caso, quindi non è necessario aggiungere ulteriori test.

Arco (2,4) → Questo arco non è coperto perché manca un test che abbia almeno un file di log e il filtro

null. Basta quindi aggiungere il test necessario (Figura 4).

Arco (3,4) → Tutti i test che hanno almeno un file di log e un filtro non null coprono questo caso, quindi non è necessario aggiungere ulteriori test.

È stato necessario quindi aggiungere un caso di test e quindi la attuale test suite è descritta nella Tabella 5

Ora andiamo a valutare la coverage ottenuta con jacoco con i casi di test definiti fino ad ora (Figura 2). Come si può vedere dalla figura 2, la statement coverage è del 100% (colonna missed instruction), e quindi tutte le linee di codice del metodo sono state eseguite.

Invece la condition coverage (colonna missed branches) è del 83%, ed è quindi migliorabile. Andando a vedere in dettaglio il report, si può notare che una condizione che non è stata coperta è una delle 4 combinazioni dell'if a riga 106 come si vede dalla Figura 3. La condizione presa in esame è la seguente:

```
logFiles == null || logFiles.length == 0
```

il branch mancato è quello dove logFiles == null, questo perchè se il parametro di input journalDir è null non esegue proprio il metodo listFiles(). Questo è dovuto al fatto che il path non era esistente e quindi il metodo tornava null. Andiamo quindi a definire un caso di test che copra questo branch. Per farlo andiamo semplicemente basta aggiungere un category partition che copre il caso di una cartella esistente ma vuota (Tabella 6). Ora quindi l'insieme dei test è quello mostrato nella tabella 7, e grazie a questo test la condition coverage è salita al 91% (Figura 4), con un aumento del 8%.

2.1.5 Adequacy Data Flow

Andiamo ora ad fare un lavoro di adeguatezza del dataflow utilizzando il framework badua per calcolare la all-uses coverage.

Dalla Figura 6 possiamo calcolare la all uses coverage.

$$CU_c + PU_c = 18$$

$$(CU + PU) - (CU_f + PU_f) = 18 + 2 = 20$$

$$\frac{CU_c + PU_c}{(CU + PU) - (CU_f + PU_f)} = \frac{18}{20} = 0.9$$

Ci sono quindi 2 coppie def-use che non sono state coperte, e quindi la all uses coverage è del 90%. Per aumentare la coverage è necessario aggiungere un caso di test che copra almeno una di queste coppie. Prendiamo in considerazione la coppia def-use sulla variabile filter non coperta. Si tratta di una coppia c-use dove il target è il continuo del ciclo for in cui è contenuto lo statement. In effetti non c'è nessun caso di test che va a coprire il caso in cui la coppia def-use è coperta e successivamente continua il ciclo. Per fare ciò vado ad aggiungere un caso di test che continua il ciclo for una volta rifiutata l'espressione booleana in cui è contenuto il comando. Per fare ciò basta aggiungere una cartella in cui controlla un altro id di journal dopo averne rifiutato uno. Effettivamente badua è servito a migliorare la qualità dei test perchè nonostante la coverage coprisse già queste righe di codice non era stata in grado di intercettare la mancanza di test su questo comportamento.

Il caso di test è descritto nella Tabella 8. Quello che ho fatto è creare una nuova cartella che contiene due file di log e un filtro con condizione >1, il primo dei due file viene rifiutato e il secondo invece viene accettato. In questo modo il metodo dopo aver scartato il primo file di log, continua con la sua ricerca e trova il secondo file di log che viene accettato. In questo modo la coppia def-use sulla variabile filter viene coperta e la all uses coverage sale (Figura 7). Con questo caso di test vado a coprire anche l'altra coppia def-use che non era stata coperta. Questo succede perchè anche a lei mancava il caso in cui una volta che il filtro rifiutava un id il metodo continuava la sua ricerca e trovava un id successivo. Così la all-use coverage riesce a raggiungere il 100%. Inoltre come effetto secondario si può notare che anche la condition coverage è salita al 100% (Figura 8)

2.1.6 Mutation Testing

Andiamo ora a fare un lavoro di adeguatezza utilizzando il framework pitest per calcolare la mutation coverage. Come si può notare dalla Figura 9 vengono rilevate tutte le mutazioni tranne 1. Quindi

possiamo calcolare la mutation coverage come segue:

$$\frac{Mutazionirilevate}{Mutazionitotali} = \frac{6}{7} = 0.85$$

Prendiamo in considerazione quindi la mutazione a riga 125, che non viene rilevata dalla test suite. Si tratta di una mutazione che omette la riga che si occupa di ordinare la lista di file di log.

Per aumentare la coverage vado quindi a modificare la test suite per coprire anche la mutazione a riga 125. Siccome la copertura manca perchè nei test correnti non viene mai ritornato un log con dimensione >1 , e quindi non c'è niente da ordinare. In particolare nel test che andrò ad aggiungere andrò a creare una cartella con 2 file di log che hanno id decrescente, in questo modo il metodo ordina i file di log e quindi la mutazione viene rilevata. Tuttavia ciò non è possibile perchè quando si vanno ad inserire dei file in una cartella essi vengono ordinati già in ordine alfabetico e quindi quando vengono presi sono già ordinati. Non è quindi possibile naturalmente creare una situazione in cui i file che vengono presi dalla cartella tramite il metodo `listFiles` non siano già ordinati.

Quindi ci sono due strade da poter percorrere:

- Andare ad etichettare l'implementazione del metodo senza la riga 112 (la riga che si occupa di ordinare la lista) come equivalente a quella senza la riga, e quindi non considerare la mutazione valida perchè è un codice che non comporta nessun cambiamento.
- Ipotizzare che in futuro possa esserci un caso in cui la lista non è ordinata, dovuto ad un cambiamento del codice della libreria, oppure ad un sistema operativo diverso. Quindi andare a creare una situazione impossibile tramite mock per verificare questa eventualità.

Ho scelto di intraprendere la seconda strada perchè mi sembra la più manutenibile, vado quindi a mockare il metodo `listFiles`, in modo tale che una volta aver recuperato i file li vada a ordinare in modo decrescente (Figura 12), in modo da simulare il comportamento non altrimenti possibile nelle condizioni attuali.

Ora effettivamente la mutazione viene rilevata (Figura 9), e quindi la mutazione viene rilevata. Quindi tutte le mutazioni sono state rilevate e quindi la mutation coverage è del 100%

2.1.7 Reliability

Considero come profili operazionali gli input della test suite sviluppata perchè essi sono derivanti dalle classi di equivalenza e quindi sono la migliore rappresentazione a mia disposizione dei possibili input del sistema. Inoltre non ho documentazione che mi permette di concludere che un caso sia più o meno probabile di un altro e quindi li considero equiprobabili. Date queste assunzioni e siccome questi test non rilevano nessuna failure del sistema, la reliability in questo caso è 1. Ciò, tuttavia, non significa che non siano presenti bug che la mia test suite non è stata in grado di rilevare.

2.2 FileInfo

2.2.1 Descrizione

Path: `org.apache.bookkeeper.bookie.FileInfo.java`

Metodo considerato: `readHeader`

Descrizione metodo: Legge l'header di un ledger index file per verificarne la correttezza. Lancia una eccezione se l'header non è corretto, altrimenti non ritorna nulla.

Come input prende ovviamente il file di cui deve leggere l'header. Tuttavia per rendere la category partition più sensata vado a considerare l'input come le componenti di cui è formato l'header, ovvero:

Input:

- magic bytes
- len of master key
- master key

2.2.2 Category Partition

Magic Bytes: {valido}, {non valido}, {null}

Master key: {null}, {master key di 0 byte}, {master key di >0 byte}

Len of master key: { = len master key } { < len master key } { > len master key } { null }

Siccome in documentazione non c'è scritto niente su questo stato dell'header, ho abbandonato momentaneamente l'approccio black box sulla category partition di questo parametro, e andando a controllare nel codice ho scoperto che nella classe viene solamente controllato se lo stato è fenced o non fenced, quindi ho descritto la category partition in questo modo:

state: {null} {fenced} {non fenced}

2.2.3 Boundary Analysis

Andiamo a definire per ogni partizione i Boundary values (Figura 10 e Figura 11). Nell'header c'è un campo che indica la versione, non specificato nella documentazione, ma analizzando il codice si può notare che ne esistono due tipi, la versione 0 e la versione 1. Per questo motivo vado a specificare anche una category partition e boundary analysis per questo campo, in modo tale da poter testare entrambe le versioni.

Andrò ad utilizzare un approccio unidimensionale nella stesura dei casi di test, tranne per le coppie di parametri master key e len of master key, che essendo strettamente correlate ho ritenuto opportuno testare tutte le combinazioni tra loro. I casi di test sono specificati nella Tabella 12. Codice implementato Figura 14.

Andando ad eseguire questi test molti di questi falliscono (Figura 15), andiamo a documentare perché reputo dei bug i fallimenti di questi test.

Il primo test che fallisce è il numero 1. Fallisce perché ho ritenuto opportuno che se il campo dell'header che indica la lunghezza della master key mandi un'eccezione nel caso in cui non coincida con la lunghezza effettiva della masterkey inserita, invece il metodo non si accorge di niente. Anche i test 2 e 5 falliscono per la stessa motivazione. Invece per quanto riguarda il test 3, questo fallisce perché ritorna un'eccezione anche se non dovrebbe. L'eccezione descrive il fatto che -1 non è una lunghezza accettata per la masterkey, tuttavia nella documentazione è esplicitamente specificato che -1 indica il fatto che non viene specificata nessuna master key, esattamente come viene fatto nel caso di test considerato. Nonostante questi casi falliscano per andare avanti con l'analisi e per non falsare i dati di coverage calcolati dai framework li andrò a modificare facendoli passare (anche se non dovrebbero).

2.2.4 Adequacy Control Flow

Ora per verificare l'adeguatezza dei casi di test, vado ad utilizzare Jacoco per calcolare la coverage. Come si può vedere della figura 13 con questa test suite otteniamo una statement coverage del 61% e una condition coverage del 59%.

Andando ad analizzare in maniera più specifica quali sono le parti non coperte (Figura 16) si può notare a riga 243 che se si usa la versione V1 dell'header è possibile specificare una parte aggiuntiva dell'header chiamata explicitLacBufLength.

Sono andato quindi ad aggiungere un valore di input ai test e due casi di test per coprire questo input aggiuntivo.

Category Partition explicitLacBufLength → { >= LAC_METADATA_LENGTH },
{ < LAC_METADATA_LENGTH AND >= 0 }, { < 0 }

Essendo il campo specificato come length, mi è sembrato opportuno descrivere anche il caso in cui la lunghezza è negativa. Nel caso d'uso del programma LAC_METADATA_LENGTH è impostato in maniera finale come 16, quindi andando a fare una boundary analysis su questo parametro ottengo i seguenti valori:

explicitLacBufLength

- { >= 16 } = 16

- $\{ <16 \text{ AND } \geq 0 \} = 15, 0$
- $\{ <0 \} = -1$

La test suite ora è composta in questo modo: (Tabella 13).

Grazie a questo incremento la statement coverage è salita al 82% e la condition coverage al 77% (Figura 17)

2.2.5 Adequacy Data Flow

Andiamo ora a fare un lavoro di adeguatezza del dataflow utilizzando il framework badua per calcolare la all-uses coverage.

Dalla Figura 18 possiamo calcolare la all uses coverage.

$$CU_c + PU_c = 66$$

$$(CU + PU) - (CU_f + PU_f) = 66 + 16 = 82$$

$$\frac{CU_c + PU_c}{(CU + PU) - (CU_f + PU_f)} = \frac{66}{82} = 0.8$$

Prendendo in considerazione le coppie def-use in Figura 19 possiamo notare che effettivamente la test suite non copre la coppia def-use sulla variabile version dalla riga 226 alla riga 228. Andiamo quindi a generare un caso di test che copra questa coppia def-use.

Per fare ciò andiamo a modificare il test che copre la versione 0 dell'header, aggiungendo un caso in cui la versione è 2 (Tabella 14). Rifacendo la build con Badua si può notare che la coppia def-use ora risulta coperta (Figura 20) e c'è stato un incremento delle coppie def-use coperte (Figura 21). Ricalcolando la all-uses coverage $CU_c + PU_c = 69$

$$(CU + PU) - (CU_f + PU_f) = 69 + 12 = 82$$

$$\frac{CU_c + PU_c}{(CU + PU) - (CU_f + PU_f)} = \frac{69}{82} = 0.84$$

si può notare che la all-uses coverage è salita al 84%

2.2.6 Mutation Testing

Vado ora a fare un analisi delle mutazioni utilizzando il framework pitest.

Come si può vedere dalla Figura 22 la mutation coverage è del 10% in riferimento all'intera classe. Se invece andiamo a vedere in dettaglio il metodo readHeader (Figura 23), la mutation coverage è

$$\frac{Mutazionirilevate}{Mutazionitotali} = \frac{15}{16} = 0.93$$

Prendiamo in considerazione per migliorare la test suite la mutazione non rilevata a riga 235. La mutazione (Figura 24) trasforma il $>$ in \geq .

Per risolvere il problema basterebbe passare come lunghezza della master key un valore che equivale al numero di byte rimanenti del file. In questo modo la mutazione verrebbe rilevata perchè si riuscirebbe a distinguere il caso in cui è $>$ da quello in cui è \geq . Tuttavia la mutazione non viene rilevata perchè anche se la lunghezza della master key dichiarata fosse uguale alla lunghezza rimanente, l'header contiene degli altri campi che vengono letti dopo la master key che generano la stessa eccezione che genererebbe il codice in quel caso. Quindi non è possibile rilevare la mutazione in questo modo. Inoltre il controllo a riga 235 con relativo lancio dell'eccezione è superfluo, perchè quando successivamente si tenta la lettura se la lunghezza dei byte che si vogliono leggere è maggiore della lunghezza rimanente del file, viene lanciata la stessa eccezione già dalla libreria di java. Quindi posso considerare il codice equivalente anche senza questo blocco di codice.

In questo modo posso considerare la mutation coverage del 100%.

2.2.7 Reliability

Considero come profili operazionali gli input della test suite sviluppata perchè essi sono derivanti dalle classi di equivalenza e quindi sono la migliore rappresentazione a mia disposizione dei possibili input del sistema. Inoltre non ho documentazione che mi permette di concludere che un caso sia più o meno probabile di un altro e quindi li considero equiprobabili. Date queste assunzioni nella mia test suite sono risultate 4 failure su un totale di 11 profili operazionali, quindi la reliability è del 63%.

3 Avro

3.1 Schema

3.1.1 Descrizione

La classe Schema è una classe astratta che rappresenta uno schema Avro. E' una delle classi core di Avro, e viene utilizzata per rappresentare i dati serializzati.

Path: org.apache.avro.Schema.java

Metodo considerato: createRecord()

Descrizione metodo: che si occupa di creare uno schema di tipo record

Input: name, doc, namespace, isError, fields

Output: Schema

La documentazione è in Figura 25.

3.1.2 Category Partition

La category partition del metodo createRecord è descritta nella tabella 15. Il campo name rappresenta il nome dello schema e siccome è obbligatorio ho partizionato in modo da testare questa caratteristica. Stessa discorso ma speculare vale per il campo doc e namespace, dato che sono descritti come opzionali quindi andranno testati in maniera differente. Per quanto riguarda il campo isError, esso non è descritto nella documentazione, quindi essendo un campo booleano ho deciso di partizionarlo in modo da testare entrambi i valori.

Il campo fields essendo un vettore di elementi, ho deciso di partizionarlo in modo da testare i casi in cui il vettore è vuoto, contiene un solo elemento e contiene più di un elemento. Ho deciso di non addentrarmi nella specifica di questi elementi, in quanto sarebbe fuori contesto nell'analisi di questo unit test.

3.1.3 Boundary Analysis

La Boundary Analysis è descritta nella tabella 16. Da essa ricavo la test suite (Tabella 17). Il motivo per cui i test 3 e 4 devono restituire una eccezione è perchè nella documentazione è specificato che il campo name è obbligatorio, e quindi se non viene specificato deve essere lanciata una eccezione. Invece il motivo per cui il 5 e 6 test devono restituire una eccezione è perchè nella documentazione è specificato che fields è obbligatorio, e quindi se non viene specificato deve essere lanciata una eccezione.

Come si vede dalla Figura 26 i test 4 e 6 falliscono. Significa che il metodo non si accorge che il campo name e fields non sono stati specificati.

Vado a modificare i due metodi facendoli passare (anche se non dovrebbero) per poter andare avanti con l'analisi e per non falsare i dati di coverage calcolati dai framework.

3.1.4 Adequacy Control Flow

Andiamo ora a valutare la coverage ottenuta con jacoco con i casi di test definiti. Per farlo consideriamo il fatto che in realtà il metodo createRecord() è solo un wrapper di una new della classe interna RecordSchema, quindi per calcolare la coverage dobbiamo prendere in considerazione il costruttore della classe RecordSchema.

Però la vera logica del metodo è contenuta nel metodo privato setField(). Quindi per fare considerazioni sulla coverage dobbiamo prendere in considerazione l'insieme di questi metodi.

Dalla Figura 27 si può vedere che la statement coverage della classe è del 12%. Tuttavia come spiegato in precedenza il metodo createRecord() è solo un wrapper di una new della classe interna RecordSchema che invece ha una statement coverage del 17% (Figura 28). Inoltre il costruttore e il metodo setField() che rappresenta la vera logica del metodo preso in esame hanno rispettivamente una coverage del 100% e del 59% (Figura 29).

Quindi per migliorare i miei casi di test andrò ad analizzare principalmente il metodo `setFied()` con un approccio white box. Data la situazione in Figura 30 vado ad incrementare l'adeguatezza dei casi di test andando a coprire il caso in cui il field ha il campo `position = -1`. Per farlo dato che per farlo controllava questo branch tramite un attributo privato ho fatto in modo di settare questo attributo a -1 tramite reflection. Quindi ho aggiunto un caso di test che copre questo caso nella test suite (Tabella 18). Con questo incremento ho ottenuto una coverage del 70% (Figura 31).

3.1.5 Adequacy Data Flow

Vado ora a fare un lavoro di adeguatezza del dataflow utilizzando il framework badua per calcolare la all-uses coverage.

Dalla Figura 32 possiamo calcolare la all uses coverage.

$$CU_c + PU_c = 25$$

$$(CU + PU) - (CU_f + PU_f) = 33$$

$$\frac{CU_c + PU_c}{(CU + PU) - (CU_f + PU_f)} = \frac{25}{33} = 0.75$$

Per migliorare la all-uses coverage andiamo a coprire la coppia def-use (956, 959) della variabile `existingField` che non è stata coperta dai precedenti test. Questa coppia indica la mancanza del caso in cui due field uguali vengono inseriti, quindi per coprire la coppia def-use andiamo ad aggiungere un test che manda in input due field uguali (Tabella 19). Con questo incremento la all-uses coverage è salita al 93% (Figura 33).

3.1.6 Mutation Testing

Vado ora a fare un analisi delle mutazioni utilizzando il framework pitest.

Come si può vedere dalla Figura 34 la mutation coverage è del 4% in riferimento all'intera classe. Se invece andiamo a vedere in dettaglio il metodo, possiamo lavorare con una mutation coverage più a grana fine (Figura 35, 36, 37). In particolare posso calcolare questa mutation coverage come segue:

$$\frac{\text{Mutazionirilevate}}{\text{Mutazionitotali}} = \frac{4}{6} = 0.66$$

Prendiamo in considerazione per migliorare la test suite la mutazione non rilevata a riga 224 (Figura 35, Figura 38). Questa mutazione non viene rilevata perchè fino ad adesso nella test suite ho considerato come expected value o un eccezione o un successo (che indica semplicemente una non eccezione), quindi il caso in cui viene ritornato null non viene rilevato. Per risolvere il problema basta modificare la test suite aspettando come valore di ritorno un oggetto di tipo Schema nei casi di successo (Tabella 20).

Con questo incremento la mutazione viene rilevata (Figura 39), e quindi la mutation coverage diventa dell' 83%.

3.1.7 Reliability

Considero come profili operazionali gli input della test suite sviluppata perchè essi sono derivanti dalle classi di equivalenza e quindi sono la migliore rappresentazione a mia disposizione dei possibili input del sistema. Inoltre non ho documentazione che mi permette di concludere che un caso sia più o meno probabile di un altro e quindi li considero equiprobabili. Date queste assunzioni nella mia test suite sono risultate 2 failure su un totale di 8 profili operazionali, quindi la reliability è del 75%.

3.2 SchemaCompatibility

3.2.1 Descrizione

SchemaCompatibility

Path: org.apache.avro.SchemaCompatibility.java

Metodo considerato: checkReaderWriterCompatibility

Per testare questa classe mi concentrerò sul metodo esposto checkReaderWriterCompatibility che si occupa di verificare la compatibilità tra due schemi.

Partiamo dalla definizione di compatibilità descritta dalla documentazione:

Evaluate the compatibility between a reader schema and a writer schema. A reader and a writer schema are declared compatible if all datum instances of the writer schema can be successfully decoded using the specified reader schema.

Input: reader, writer

Output: boolean.

In realtà il metodo java effettivo non ritorna un boolean ma un oggetto con una descrizione che segnala se è compatibile o meno. Ma dato che il metodo in questione ha lo scopo semplicemente di identificare se due schemi sono compatibili ho deciso di interpretare il risultato come un booleano.

3.2.2 Category Partition

Data la descrizione di compatibilità data dalla documentazione ho interpretato che essa dipenda necessariamente dal tipo associato allo schema. Quindi ho deciso di partizionare gli input basandomi su questo criterio.

Nella documentazione è specificato che i tipi possono essere di due categorie, o primitivi o derivati.

Reader: {Schema con tipo primitivo}, {Schema con tipo derivato}

Dato che il metodo confronta due schemi per controllare la loro compatibilità, ho ritenuto necessario che la category partition di uno dipendesse dall'altro. In questo modo do importanza alla loro combinazione che ha più rilevanza ai fini del testing.

Writer: {Schema con tipo uguale al reader}, {Schema con tipo diverso dal reader}

3.2.3 Boundary Analysis

Reader: {Schema tipo int}, {Schema tipo record}

Writer: {Schema stesso tipo di Reader}, {Schema di tipo diverso da Reader ma stessa categoria}, {Schema di tipo diverso da Reader ma categoria diversa}

Test suite (Tabella 21)

3.2.4 Adequacy Control Flow

Vado ora a valutare la coverage ottenuta con jacoco con i casi di test definiti. Dalla Figura 40 si può vedere che la statement coverage della classe è del 37%. Se osserviamo solo il metodo considerato la statement coverage è del 78% e la condition coverage è del 66%. Andando a vedere in maniera più specifica quali sono le parti non coperte (Figura 41), si può osservare che la parte non coperta è semplicemente il caso di default del costrutto switch. Tuttavia non è possibile scaturire nessun caso di test che copra questo caso, in quanto non è possibile che il metodo ritorni un valore diverso nell'implementazione corrente. Quindi sarebbe opportuno mockare qualche parte per permettere di esplorare questo scenario, tuttavia non è possibile mockare nulla perchè questo comportamento è descritto solamente tramite l'uso di variabili locali che non sono raggiungibili con i mock. Quindi non è possibile aumentare la coverage di questo metodo.

Vado quindi a considerare la coverage senza la parte non raggiungibile dai casi di test, e quindi entrambe diventano del 100%

3.2.5 Adequacy Data Flow

Andiamo ora a fare un lavoro di adeguatezza del dataflow utilizzando il framework badua per calcolare la all-uses coverage.

Dalla Figura 42 possiamo calcolare la all uses coverage.

$$CU_c + PU_c = 12$$

$$(CU + PU) - (CU_f + PU_f) = 12 + 3 = 15$$

$$\frac{CU_c + PU_c}{(CU + PU) - (CU_f + PU_f)} = \frac{11}{14} = 0.8$$

Tutte le coppie use-def non coperte fanno riferimento allo stessa riga non coperta dalla statement coverage, quindi per gli stessi motivi spiegati precedentemente anche la all-uses coverage è considerabile al 100%

3.2.6 Mutation Testing

Vado ora a fare un analisi delle mutazioni utilizzando il framework pitest.

Dalla Figura 43 si può vedere che la mutation coverage è del 100% perchè pit ha generato una sola mutazione e questa è stata rilevata.

3.2.7 Reliability

Considero come profili operazionali gli input della test suite sviluppata perchè essi sono derivanti dalle classi di equivalenza e quindi sono la migliore rappresentazione a mia disposizione dei possibili input del sistema. Inoltre non ho documentazione che mi permette di concludere che un caso sia più o meno probabile di un altro e quindi li considero equiprobabili. Date queste assunzioni e siccome questi test non rilevano nessuna failure del sistema, la reliability in questo caso è 1. Ciò, tuttavia, non significa che non siano presenti bug che la mia test suite non è stata in grado di rilevare.

3.3 IT Schema-SchemaResolver

Questa sezione descrive l'implementazione di un test di integrazione che utilizza le classi Schema e SchemaResolver. Il test in questione si occupa di verificare l'integrazione tra il metodo createRecord() della classe Schema (metodo già analizzato in precedenza) e il metodo unresolvedSchema() della classe SchemaResolver.

Il test in questione va semplicemente ad utilizzare il metodo unresolvedSchema() che usa createRecord() per creare uno schema con delle caratteristiche particolari e verifica che lo schema ritornato abbia effettivamente le caratteristiche richieste.

Codice implementato in Figura 44.

Il report di Failsafe mostra che l'integration test è andato a buon fine (Figura 45).

Tabella 1: Category Partition metodo listJournalIds

journalDir	JournalIdFilter
{Directory contenente file di log}	Filtro esistente
{Directory contenente file di log e altri file}	Filtro sempre True
{Directory contenente file non di log}	Filtro sempre False
{path non esistente}	Filtro inesistente
{Path di un file}	null
null	

Tabella 2: Boundary Analysis metodo listJournalIds

journalDir	JournalIdFilter
{Directory contenente 1 file di log}	JournalRollingFilter
{Directory contenente 1 file di log e 1 file di testo}	Filtro sempre True
{Directory contenente 1 file di testo}	Filtro sempre False
{path non esistente}	MyFilter
{Path di un file di log}	null
null	

Tabella 3: Test Suite derivante dalla category partition e dalla boundary analysis

journalDir	JournalIdFilter	Risultato Atteso
Directory contenente 1 file di log	JournalRollingFilter	[1]
Directory contenente 1 file di log e 1 file di testo	Filtro sempre True	[1]
Directory contenente 1 file di testo	Filtro sempre False	[]
path non esistente	MyFilter	[]
Path di un file di log	null	Exception
null	null	Exception

Tabella 4: Test per coprire l'arco (2,4)

journalDir	JournalIdFilter	Risultato Atteso
Directory contenente 1 file di log	null	Exception

Tabella 5: Casi di test aggiornati per coprire l'arco (2,4)

journalDir	JournalIdFilter	Risultato Atteso
Directory contenente 1 file di log	JournalRollingFilter	[1]
Directory contenente 1 file di log e 1 file di testo	Filtro sempre True	[1]
Directory contenente 1 file di testo	Filtro sempre False	[]
path non esistente	MyFilter	[]
Path di un file di log	null	Exception
null	null	Exception
Directory contenente 1 file di log	null	Exception

Tabella 6: Test per aumentare la condition coverage

journalDir	JournalIdFilter	Risultato Atteso
Directory vuota	null	[]

Tabella 7: Casi di test aggiornati per aumentare la condition coverage

journalDir	JournalIdFilter	Risultato Atteso
Directory contenente 1 file di log	JournalRollingFilter	[1]
Directory contenente 1 file di log e 1 file di testo	Filtro sempre True	[1]
Directory contenente 1 file di testo	Filtro sempre False	[]
path non esistente	MyFilter	[]
Path di un file di log	null	Exception
null	null	Exception
Directory contenente 1 file di log	null	Exception
Directory vuota	null	[]

Tabella 8: Test per aumentare la all-use coverage

journalDir	JournalIdFilter	Risultato Atteso
Directory con un log con id 2 e un file con id 1	Filtro con condizione >1	[2]

Tabella 9: Test per aumentare la all-use coverage

journalDir	JournalIdFilter	Risultato Atteso
Directory con un log con id 2 e un log con id 1	Filtro sempre true	[1, 2]

Tabella 10: Boundary Analysis metodo readHeader

Magic Bytes	Master Key	Len of the master key
{valido} = BKLE {non valido} = BKLU	{master key di 0 byte} = new byte[] {master key di >0 byte} = new byte[1]	{= len effettiva} = len effettiva {<len effettiva} = len effettiva - 1 {>len effettiva} = len effettiva + 1

Tabella 11: Boundary Analysis metodo readHeader

State	Version
{fenced} = 1 {non fenced} = 0	{versione 0} = 0 {versione 1} = 1

Tabella 12: Test Suite derivante dalla category partition e dalla boundary analysis

Magic Bytes	Master key	Len of master key	State	Version	Risultato Atteso
BKLE	new byte[1]	1	0	1	Success
BKLE	new byte[1]	2	0	1	Exception
BKLE	new byte[1]	0	0	1	Exception
BKLE	new byte[0]	-1	0	0	Success
BKLE	new byte[0]	0	0	0	Success
BKLE	new byte[0]	1	0	0	Exception
BKLU	new byte[1]	1	0	1	Exception

Tabella 13: Test Suite derivante dall'incremento dopo Jacoco

Magic Bytes	Master key	Len of master key	State	Version	BufLength	Risultato Atteso
BKLE	new byte[1]	1	0	1	0	Success
BKLE	new byte[1]	2	0	1	0	Exception
BKLE	new byte[1]	0	0	1	0	Exception
BKLE	new byte[0]	-1	0	0	0	Success
BKLE	new byte[0]	0	0	0	0	Success
BKLE	new byte[0]	1	0	0	0	Exception
BKLU	new byte[1]	1	0	1	0	Exception
BKLE	new byte[1]	1	0	1	16	Success
BKLE	new byte[1]	1	0	1	15	Exception
BKLE	new byte[1]	1	0	1	-1	Exception

Tabella 14: Test Suite derivante dall'incremento dopo Badua

Magic Bytes	Master key	Len of master key	State	Version	BufLength	Risultato Atteso
BKLE	new byte[1]	1	0	1	0	Success
BKLE	new byte[1]	2	0	1	0	Exception
BKLE	new byte[1]	0	0	1	0	Exception
BKLE	new byte[0]	-1	0	0	0	Success
BKLE	new byte[0]	0	0	0	0	Success
BKLE	new byte[0]	1	0	0	0	Exception
BKLU	new byte[1]	1	0	1	0	Exception
BKLE	new byte[1]	1	0	1	16	Success
BKLE	new byte[1]	1	0	1	15	Exception
BKLE	new byte[1]	1	0	1	-1	Exception
BKLE	new byte[1]	1	0	2	-1	Exception

Tabella 15: Category Partition metodo createRecord

Category Partition CreateRecord				
name	doc	namespace	isError	fields
{null}	{null}	{null}	{true}	{null}
{stringa vuota}	{stringa vuota}	{stringa vuota}	{false}	{array vuoto}
{stringa non vuota}	{stringa non vuota}	{stringa non vuota}		{array 1 elemento}
				{array >1 elementi}

Tabella 16: Category Partition metodo createRecord

name	doc	namespace	isError	fields
null	null	null	true	null
" "	" "	" "	false	[]
"test"	"test"	"test"		[field]
				[field1, field2]

Tabella 17: Test Suite derivante dalla category partition e dalla boundary analysis

Name	Doc	Namespace	IsError	Fields	Risultato Atteso
"test"	"test"	"test"	False	[Field1]	Success
"test"	"test"	"test"	False	[Field1, Field2]	Success
"test"	"test"	"test"	True	[Field1, Field2]	Exception
null	null	null	False	[Field1]	Exception
"test"	"test"	"test"	False	null	Exception
"test"	"test"	"test"	False	[]	Exception

Tabella 18: Test Suite derivante dall'incremento dopo Jacoco

Name	Doc	Namespace	IsError	Fields	Risultato Atteso
"test"	"test"	"test"	False	[Field1]	Success
"test"	"test"	"test"	False	[Field1, Field2]	Success
"test"	"test"	"test"	True	[Field1, Field2]	Exception
null	null	null	False	[Field1]	Exception
"test"	"test"	"test"	False	null	Exception
"test"	"test"	"test"	False	[]	Exception
"test"	"test"	"test"	False	[Field3*]	Exception

*Field3 è un field con position = -1

Tabella 19: Test Suite derivante dall'incremento dopo Badua

Name	Doc	Namespace	IsError	Fields	Risultato Atteso
"test"	"test"	"test"	False	[Field1]	Success
"test"	"test"	"test"	False	[Field1, Field2]	Success
"test"	"test"	"test"	True	[Field1, Field2]	Exception
null	null	null	False	[Field1]	Exception
"test"	"test"	"test"	False	null	Exception
"test"	"test"	"test"	False	[]	Exception
"test"	"test"	"test"	False	[Field3*]	Exception
"test"	"test"	"test"	False	[Field1, Field1]	Exception

*Field3 è un field con position = -1

Tabella 20: Test Suite derivante dall'incremento dopo Pit

Name	Doc	Namespace	IsError	Fields	Risultato Atteso
"test"	"test"	"test"	False	[Field1]	Object(Schema)
"test"	"test"	"test"	False	[Field1, Field2]	Object(Schema)
"test"	"test"	"test"	True	[Field1, Field2]	Exception
null	null	null	False	[Field1]	Exception
"test"	"test"	"test"	False	null	Exception
"test"	"test"	"test"	False	[]	Exception
"test"	"test"	"test"	False	[Field3*]	Exception
"test"	"test"	"test"	False	[Field1, Field1]	Exception

*Field3 è un field con position = -1

Tabella 21: Test Suite derivante dalla category partition e dalla boundary analysis

Reader	Writer	Valore atteso
Schema tipo int	Schema tipo int	True
Schema tipo Record	Schema tipo int	False
Schema tipo int	Schema tipo String	False

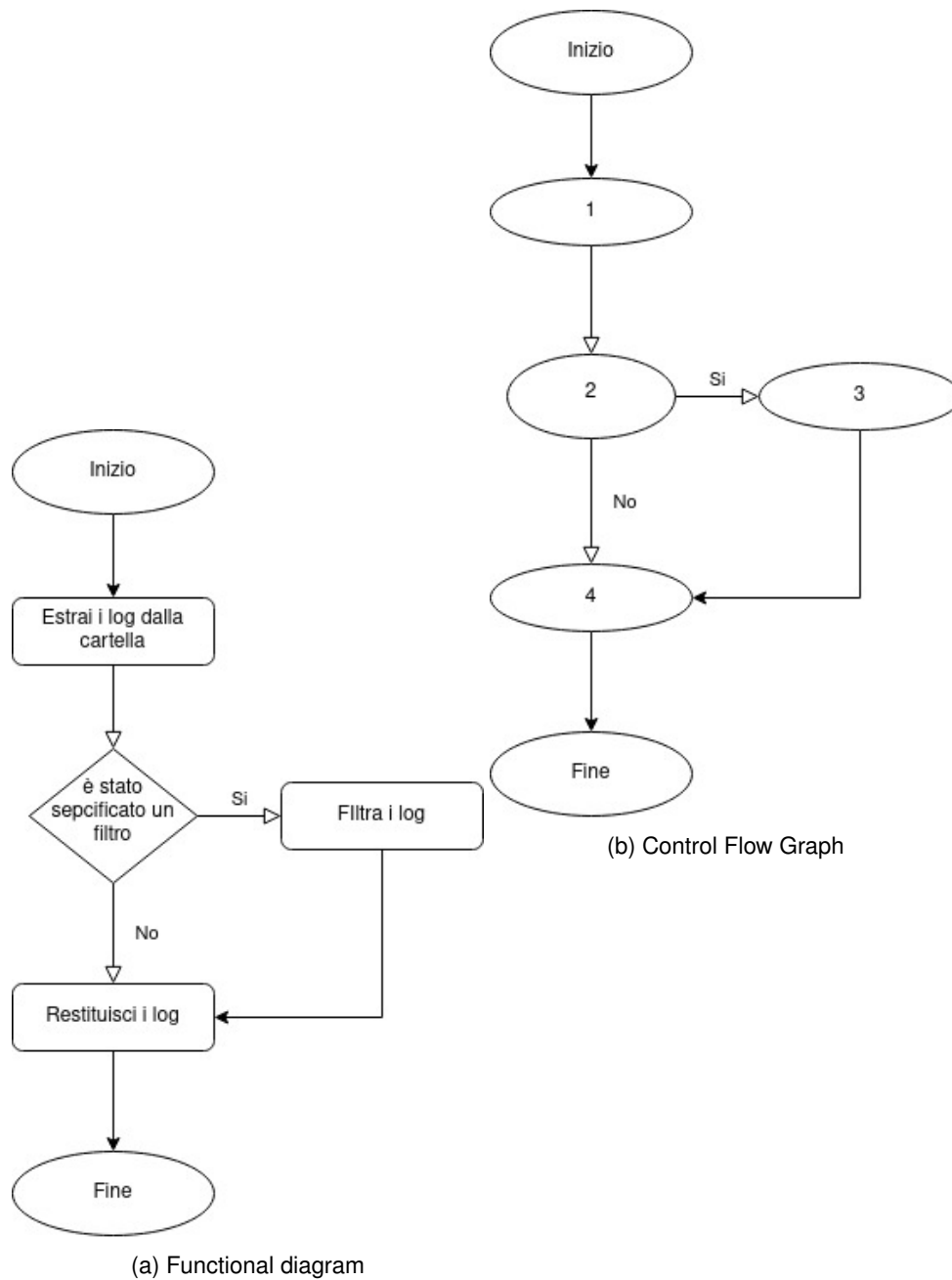


Figura 1: Diagrammi di listJournalIds

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
listJournalIds(File, Journal, JournalIdFilter)	0	100%	0	83%	2	7	0
						16	1

Figura 2: Jacoco coverage dilistJournalIds

```

103.    ~/
104.    public static List<Long> listJournalIds(File journalDir, JournalIdFilter filter) {
105.        File[] logFiles = journalDir.listFiles();
106.        if (logFiles == null || logFiles.length == 0) {
107.            return Collections.emptyList();
108.        }
109.        List<Long> logs = new ArrayList<Long>();
110.        for (File f: logFiles) {
111.            String name = f.getName();
112.            if (!name.endsWith(".txn")) {
113.                continue;
114.            }
115.            String idString = name.split("\\.")[0];
116.            long id = Long.parseLong(idString, 16);
117.            if (filter != null) {
118.                if (filter.accept(id)) {
119.                    logs.add(id);
120.                }
121.            } else {
122.                logs.add(id);
123.            }
124.        }
125.        Collections.sort(logs);
126.        return logs;
127.    }

```

Figura 3: Jacoco coverage dilistJournalIds

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
listJournalIds(File, JournalIdFilter)		100%		91%	1	7	0
						16	0
							1

Figura 4: Jacoco coverage dilistJournalIds

```

115.    @Parameterized.Parameters
116.    @Test
117.    public static Collection<Object[]> getTestParameters() {
118.        return Arrays.asList(new Object[][]{
119.            {new ArrayList<>(Collections.singletonList(1L)), null, JournalDirType.ONE_LOG_DIR.getJournalDir(), JournalIdFilterType.JOURNAL_ROLLING_FILTER.getJournalIdFilter(),
120.             {new ArrayList<>(Collections.singletonList(1L)), null, JournalDirType.ONE_LOG_DIR.getJournalDir(), null},
121.             {new ArrayList<>(Collections.singletonList(1L)), null, JournalDirType.ONE_LOG_ONE_TEXT_DIR.getJournalDir(), JournalIdFilterType.ALWAYS_TRUE_FILTER.getJournalIdFilter(),
122.             {new ArrayList<>(), null, JournalDirType.ONE_TEXT_DIR.getJournalDir(), JournalIdFilterType.ALWAYS_FALSE_FILTER.getJournalIdFilter(),
123.             {new ArrayList<>(), null, JournalDirType.NOT_EXISTING_DIR.getJournalDir(), JournalIdFilterType.NEW_FILTER.getJournalIdFilter(),
124.             {new ArrayList<>(), null, JournalDirType.LOG_FILE.getJournalDir(), null},
125.             {null, Exception.class, null, null},
126.         }});
127.    }
128.    @Before
129.    public void setUp() {
130.        MockitoAnnotations.initMocks(this);
131.        if (journalDir != null && journalDir.getPath().substring(beginIndex: journalDir.getPath().length() - 8).equals("void_dir")) {
132.            boolean succes = journalDir.mkdir();
133.            if (!succes) {
134.                System.out.println("Could not create void dir");
135.            } else {
136.                System.out.println("Created void dir");
137.            }
138.        }
139.    }
140.    @Test
141.    public void getJournalIdsTest() {
142.        try {
143.            System.out.println("Journal dir: " + journalDir.getPath());
144.            System.out.println("Journal dir list files: " + Arrays.toString(journalDir.listFiles()));
145.            List<Long> journalIds = Journal.listJournalIds(journalDir, journalIdFilter);
146.            System.out.println("Result: " + journalIds);
147.            System.out.println("Expected result: " + expectedResult);
148.            assertEquals(journalIds, expectedResult);
149.        } catch (Exception e) {
150.            System.out.println("Exception: " + e);
151.            if (expectedException != null && expectedException.isAssignableFrom(e.getClass())) {
152.                assertEquals(true);
153.            } else {
154.                assertEquals(false);
155.            }
156.        }
157.    }
158.    }

```

Figura 5: Codice sorgente di JournalTest

```

<class name="org/apache/bookkeeper/bookie/Journal">
  <method name="listJournalIds" desc="(Ljava/io/File;Lorg/apache/bookkeeper/bookie/Journal$JournalIdFilter;)Ljava/util/List;">
    <du var="filter" def="105" use="117" target="118" covered="1"/>
    <du var="filter" def="105" use="117" target="122" covered="1"/>
    <du var="filter" def="105" use="118" target="119" covered="1"/>
    <du var="filter" def="105" use="118" target="110" covered="0"/>
    <du var="logFiles" def="105" use="106" target="106" covered="1"/>
    <du var="logFiles" def="105" use="106" target="107" covered="1"/>
    <du var="logFiles" def="105" use="106" target="107" covered="1"/>
    <du var="logFiles" def="105" use="106" target="109" covered="1"/>
    <du var="logFiles" def="105" use="110" covered="1"/>
    <du var="logs" def="109" use="125" covered="1"/>
    <du var="logs" def="109" use="126" covered="1"/>
    <du var="logs" def="109" use="122" covered="1"/>
    <du var="logs" def="109" use="119" covered="1"/>
    <du var="name" def="111" use="112" target="113" covered="1"/>
    <du var="name" def="111" use="112" target="115" covered="1"/>
    <du var="name" def="111" use="115" covered="1"/>
    <du var="id" def="116" use="122" covered="1"/>
    <du var="id" def="116" use="118" target="119" covered="1"/>
    <du var="id" def="116" use="118" target="110" covered="0"/>
    <du var="id" def="116" use="119" covered="1"/>
    <counter type="DU" missed="3" covered="28"/>
    <counter type="METHOD" missed="0" covered="1"/>
  </method>

```

Figura 6: Badua coverage dilistJournalIds

```

<class name="org/apache/bookkeeper/bookie/Journal">
  <method name="listJournalIds" desc="(Ljava/io/File;Lorg/apache/bookkeeper/bookie/Journal$JournalIdFilter;)Ljava/util/List;">
    <du var="filter" def="105" use="117" target="118" covered="1"/>
    <du var="filter" def="105" use="117" target="122" covered="1"/>
    <du var="filter" def="105" use="118" target="119" covered="1"/>
    <du var="filter" def="105" use="118" target="110" covered="1"/>
    <du var="logFiles" def="105" use="106" target="106" covered="1"/>
    <du var="logFiles" def="105" use="106" target="107" covered="1"/>
    <du var="logFiles" def="105" use="106" target="107" covered="1"/>
    <du var="logFiles" def="105" use="106" target="109" covered="1"/>
    <du var="logFiles" def="105" use="110" covered="1"/>
    <du var="logs" def="109" use="125" covered="1"/>
    <du var="logs" def="109" use="126" covered="1"/>
    <du var="logs" def="109" use="122" covered="1"/>
    <du var="logs" def="109" use="119" covered="1"/>
    <du var="name" def="111" use="112" target="113" covered="1"/>
    <du var="name" def="111" use="112" target="115" covered="1"/>
    <du var="name" def="111" use="115" covered="1"/>
    <du var="id" def="116" use="122" covered="1"/>
    <du var="id" def="116" use="118" target="119" covered="1"/>
    <du var="id" def="116" use="118" target="110" covered="1"/>
    <du var="id" def="116" use="119" covered="1"/>
    <counter type="DU" missed="1" covered="30"/>
    <counter type="METHOD" missed="0" covered="1"/>
  </method>

```

Figura 7: Badua coverage dilistJournalIds

Journal

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
listJournalIds(File, Journal.JournalIdFilter)	0	100%	0	100%	0	7	0

Figura 8: Jacoco coverage dilistJournalIds

```

104 public static List<Long> listJournalIds(File journalDir, JournalIdFilter filter) {
105     File[] logFiles = journalDir.listFiles();
106     if (logFiles == null || logFiles.length == 0) {
107         return Collections.emptyList();
108     }
109     List<Long> logs = new ArrayList<Long>();
110     for (File f: logFiles) {
111         String name = f.getName();
112         if (!name.endsWith(".txn")) {
113             continue;
114         }
115         String idString = name.split("\\.")[0];
116         long id = Long.parseLong(idString, 16);
117         if (filter != null) {
118             if (filter.accept(id)) {
119                 logs.add(id);
120             }
121             } else {
122                 logs.add(id);
123             }
124         }
125     Collections.sort(logs);
126     return logs;
127 }

```

Figura 9: Pitest coverage dilistJournalIds

```

104 public static List<Long> listJournalIds(File journalDir, JournalIdFilter filter) {
105     File[] logFiles = journalDir.listFiles();
106     if (logFiles == null || logFiles.length == 0) {
107         return Collections.emptyList();
108     }
109     List<Long> logs = new ArrayList<Long>();
110     for (File f: logFiles) {
111         String name = f.getName();
112         if (!name.endsWith(".txn")) {
113             continue;
114         }
115         String idString = name.split("\\.")[0];
116         long id = Long.parseLong(idString, 16);
117         if (filter != null) {
118             if (filter.accept(id)) {
119                 logs.add(id);
120             }
121             } else {
122                 logs.add(id);
123             }
124         }
125     Collections.sort(logs);
126     return logs;
127 }

```

Figura 10: Mutazioni rilevate da Pitest

```

public static List<Long> listJournalIds(File journalDir, JournalIdFilter filter) {
    File[] logFiles = journalDir.listFiles();
    if (logFiles == null || logFiles.length == 0) {
        return Collections.emptyList();
    }
    List<Long> logs = new ArrayList<>();
    for (File f: logFiles) {
        String name = f.getName();
        if (name.endsWith(".txn")) {
            continue;
        }
        String idString = name.split(regex: "\\.").[0];
        long id = Long.parseLong(idString, radix: 16);
        if (filter != null) {
            if (filter.accept(id)) {
                logs.add(id);
            } else {
                logs.add(id);
            }
        }
    }
    Collections.sort(logs);
    return logs;
}

```

Figura 11: Mutazione inserita nel codice sorgente di Bookkeeper

```

127  // luca
128  @Parameterized.Parameters
129  public static Collection<Object[]> getTestParameters() {
130      return Arrays.asList(new Object[][]{
131          {new ArrayList<>(Collections.singletonList(1L)), null, JournalDirType.ONE_LOG_DIR.getJournalDir(), JournalIdFilterType.JOURNAL_ROLLING_FILTER.getJournalIdFilter(),
132            {new ArrayList<>(Collections.singletonList(1L)), null, JournalDirType.ONE_LOG_DIR.getJournalDir(), null},
133            {new ArrayList<>(Collections.singletonList(1L)), null, JournalDirType.ONE_LOG_DIR.getJournalDir(), JournalIdFilterType.ALWAYS_TRUE_FILTER.getJournalIdFilter(),
134            {new ArrayList<>(Collections.singletonList(1L)), null, JournalDirType.ONE_LOG_DIR.getJournalDir(), JournalIdFilterType.ALWAYS_FALSE_FILTER.getJournalIdFilter(),
135            {new ArrayList<>(), null, JournalDirType.NOT_EXISTING_DIR.getJournalDir(), JournalIdFilterType.ALWAYS_FALSE_FILTER.getJournalIdFilter(),
136            {new ArrayList<>(), null, JournalDirType.LOG_FILE.getJournalDir(), null},
137            {null, Exception.class, null, null},
138            {new ArrayList<>(), null, JournalDirType.VOID_DIR.getJournalDir(), null},
139            {new ArrayList<>(Collections.singletonList(2L)), null, JournalDirType.ADF_DIR.getJournalDir(), JournalIdFilterType.ADF_FILTER.getJournalIdFilter(),
140            {new ArrayList<>(Arrays.asList(1L, 2L)), null, JournalDirType.ADF_DIR.getJournalDir(), JournalIdFilterType.ALWAYS_TRUE_FILTER.getJournalIdFilter()}}
141      });
142  // luca
143  @Before
144  public void setUp() {
145      MockitoAnnotations.initMocks(this);
146      if (journalDir != null && journalDir.getPath().substring(beginIndex, journalDir.getPath().length() - 8).equals("void_dir")) {
147          boolean succes = journalDir.mkdir();
148          if (!succes) {
149              System.out.println("Could not create void dir");
150          } else {
151              System.out.println("Created void dir");
152          }
153      }
154      if (journalDir != null && journalDir.getPath().equals(JournalDirType.ADF_DIR.getJournalDir().getPath())) {
155          File[] output = journalDir.listFiles();
156          reverse(output);
157          System.out.println("reversed output:\n" + Arrays.toString(output));
158          journalDir = Mockito.spy(journalDir);
159          doReturn(output).when(journalDir).listFiles();
160      }
161  }
162  // luca

```

Figura 12: Codice sorgente di JournalTest aggiornato con il mock

Element	Missed Instructions	Cov. %	Missed Branches	Cov. %	Missed Cxty	Missed Lines	Missed Methods
● readHeader()	<div><div></div></div>	61%	<div><div></div></div>	59%	7 12	13 41	0 1

Figura 13: Jacoco coverage readHeader

```

@Parameterized.Parameters
public static Collection<Object[]> data(){
    return Arrays.asList(new Object[][]{
        {null, ByteBuffer.wrap("BKLE".getBytes(UTF_8)).getInt(), new byte[1], 1, 0, 1},
        {Exception.class, ByteBuffer.wrap("BKLE".getBytes(UTF_8)).getInt(), new byte[1], 2, 0, 1},
        {Exception.class, ByteBuffer.wrap("BKLE".getBytes(UTF_8)).getInt(), new byte[1], 0, 0, 1},
        {null, ByteBuffer.wrap("BKLE".getBytes(UTF_8)).getInt(), new byte[0], -1, 0, 0},
        {null, ByteBuffer.wrap("BKLE".getBytes(UTF_8)).getInt(), new byte[0], 0, 0, 0},
        {Exception.class, ByteBuffer.wrap("BKLE".getBytes(UTF_8)).getInt(), new byte[0], 1, 1, 0},
        {Exception.class, ByteBuffer.wrap("BKLU".getBytes(UTF_8)).getInt(), new byte[1], 1, 0, 1},
    });
}

@Before
public void setUp() throws IOException {
    MockitoAnnotations.initMocks( testClass: this);
    File file = new File(PROJECT_ROOT_PATH, child: "src/test/resources/ledgers/tt.txn");
    byte[] bytes = new byte[1];
    bytes[0] = 1;
    fileInfo = new FileInfo(file, bytes, fileInfoVersionToWrite: 0);

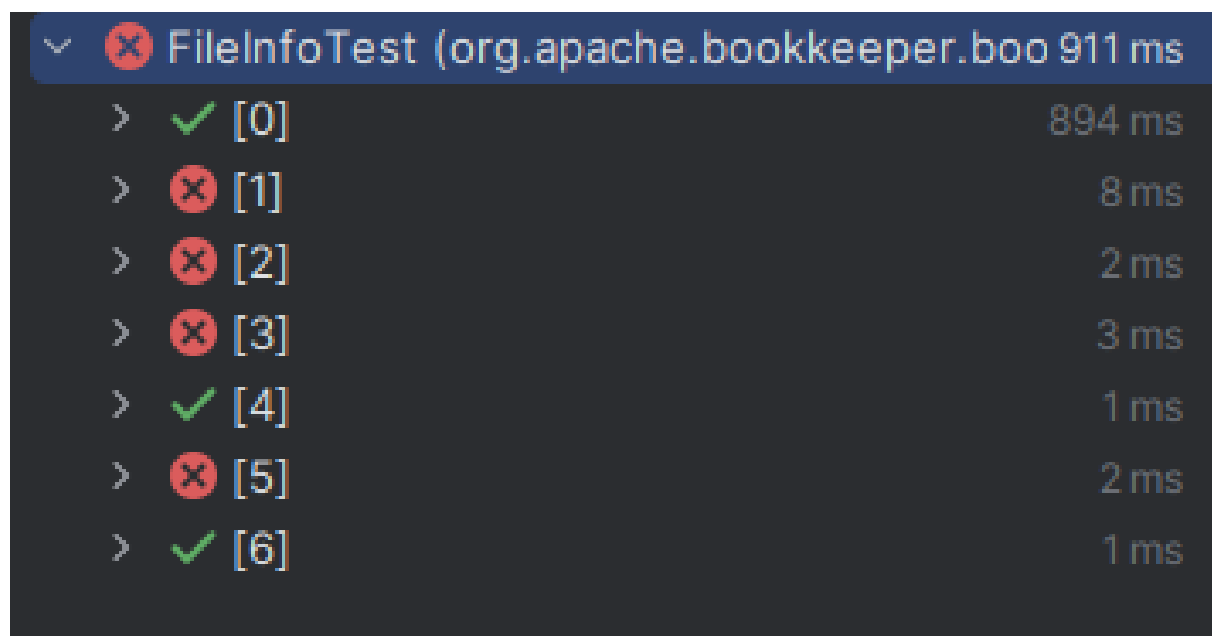
    ByteBuffer bb = ByteBuffer.allocate((int) 1024);
    bb.putInt(magicBytes);
    bb.putInt(version);
    bb.putInt(lenMasterKey);
    if(masterKey != null) {
        bb.put(masterKey);
        System.out.println(masterKey.length);
    }
    bb.putInt(state);
    bb.putInt( value: 0);

    bb.rewind();
    try(FileChannel fc = new RandomAccessFile(file, mode: "rw").getChannel()){
        fc.position( newPosition: 0);
        fc.write(bb);
    }
}

@Test
public void readHeaderTest(){
    try {
        fileInfo.readHeader();
        if(expectedException == null)
            assert true;
        else
            assert false;
    } catch (IOException e) {
        e.printStackTrace();
        if (expectedException != null && expectedException.isAssignableFrom(e.getClass())) {
            assert (true);
        } else {
            assert (false);
        }
    }
}
}

```

Figura 14: Codice sorgente di FileInfoTest



▼	✖	FileInfoTest (org.apache.bookkeeper.boo	911 ms
>	✓	[0]	894 ms
>	✖	[1]	8 ms
>	✖	[2]	2 ms
>	✖	[3]	3 ms
>	✓	[4]	1 ms
>	✖	[5]	2 ms
>	✓	[6]	1 ms

Figura 15: Fallimento test readHeader

```

207.     public synchronized void readHeader() throws IOException {
208.         if (lf.exists()) {
209.             if (fc != null) {
210.                 return;
211.             }
212.
213.             fc = new RandomAccessFile(lf, mode).getChannel();
214.             size = fc.size();
215.             sizeSinceLastWrite = size;
216.
217.             // avoid hang on reading partial index
218.             ByteBuffer bb = ByteBuffer.allocate((int) (Math.min(size, START_OF_DATA)));
219.             while (bb.hasRemaining()) {
220.                 fc.read(bb);
221.             }
222.             bb.flip();
223.             if (bb.getInt() != SIGNATURE) {
224.                 throw new IOException("Missing ledger signature while reading header for " + lf);
225.             }
226.             int version = bb.getInt();
227.             if (version > CURRENT_HEADER_VERSION) {
228.                 throw new IOException("Incompatible ledger version " + version + " while reading header for " + lf);
229.             }
230.             this.headerVersion = version;
231.
232.             int length = bb.getInt();
233.             if (length < 0) {
234.                 throw new IOException("Length " + length + " is invalid while reading header for " + lf);
235.             } else if (length > bb.remaining()) {
236.                 throw new BufferUnderflowException();
237.             }
238.             masterKey = new byte[length];
239.             bb.get(masterKey);
240.             stateBits = bb.getInt();
241.
242.             if (this.headerVersion >= V1) {
243.                 int explicitLacBufLength = bb.getInt();
244.                 if (explicitLacBufLength == 0) {
245.                     explicitLac = null;
246.                 } else if (explicitLacBufLength >= DigestManager.LAC_METADATA_LENGTH) {
247.                     if (explicitLac == null) {
248.                         explicitLac = ByteBuffer.allocate(explicitLacBufLength);
249.                     }
250.                     byte[] explicitLacBufArray = new byte[explicitLacBufLength];
251.                     bb.get(explicitLacBufArray);
252.                     explicitLac.put(explicitLacBufArray);
253.                     explicitLac.rewind();
254.                 } else {
255.                     throw new IOException("ExplicitLacBufLength " + explicitLacBufLength
256.                                             + " is invalid while reading header for " + lf);
257.                 }
258.             }
259.
260.             needFlushHeader = false;
261.         } else {
262.             throw new IOException("Ledger index file " + lf + " does not exist");
263.         }
264.     }
265.

```

Figura 16: Jacoco coverage readHeader

FileInfo

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
readHeader()	<div><div></div></div>	82%	<div><div></div></div>	77%	5	12	4	41	0	1

Figura 17: Jacoco coverage readHeader

```

<counter type="DU" missed="16" covered="65"/>
<counter type="METHOD" missed="0" covered="1"/>

```

Figura 18: Badua coverage readHeader

```

<du var="version" def="226" use="227" target="228" covered="0"/>
<du var="version" def="226" use="227" target="230" covered="1"/>
<du var="version" def="226" use="230" covered="1"/>
<du var="version" def="226" use="228" covered="0"/>

```

Figura 19: Def-use not covered

```

<du var="version" def="226" use="227" target="228" covered="1"/>
<du var="version" def="226" use="227" target="230" covered="1"/>
<du var="version" def="226" use="230" covered="1"/>
<du var="version" def="226" use="228" covered="1"/>
<du var="this.headerVersion" def="230" use="242" target="243" covered="1"/>

```

Figura 20: Def-use covered

```

<du var="this.explicitLac" def="248" use="253" covered="1"/>
<counter type="DU" missed="12" covered="69"/>
<counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 21: Badua coverage readHeader after test update

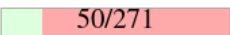
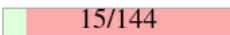
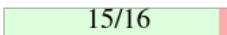
Name	Line Coverage	Mutation Coverage	Test Strength
FileInfo.java	18%  50/271	10%  15/144	94%  15/16

Figura 22: Pit coverage

```

207     public synchronized void readHeader() throws IOException {
208         if (lf.exists()) {
209             if (fc != null) {
210                 return;
211             }
212
213             fc = new RandomAccessFile(lf, mode).getChannel();
214             size = fc.size();
215             sizeSinceLastWrite = size;
216
217             // avoid hang on reading partial index
218             ByteBuffer bb = ByteBuffer.allocate((int) (Math.min(size, START_OF_DATA)));
219             while (bb.hasRemaining()) {
220                 fc.read(bb);
221             }
222             bb.flip();
223             if (bb.getInt() != SIGNATURE) {
224                 throw new IOException("Missing ledger signature while reading header for " + lf);
225             }
226             int version = bb.getInt();
227             if (version > CURRENT_HEADER_VERSION) {
228                 throw new IOException("Incompatible ledger version " + version + " while reading header for " + lf);
229             }
230             this.headerVersion = version;
231
232             int length = bb.getInt();
233             if (length < 0) {
234                 throw new IOException("Length " + length + " is invalid while reading header for " + lf);
235             } else if (length > bb.remaining()) {
236                 throw new BufferUnderflowException();
237             }
238             masterKey = new byte[length];
239             bb.get(masterKey);
240             stateBits = bb.getInt();
241
242             if (this.headerVersion >= V1) {
243                 int explicitLacBufLength = bb.getInt();
244                 if (explicitLacBufLength == 0) {
245                     explicitLac = null;
246                 } else if (explicitLacBufLength >= DigestManager.LAC_METADATA_LENGTH) {
247                     if (explicitLac == null) {
248                         explicitLac = ByteBuffer.allocate(explicitLacBufLength);
249                     }
250                     byte[] explicitLacBufArray = new byte[explicitLacBufLength];
251                     bb.get(explicitLacBufArray);
252                     explicitLac.put(explicitLacBufArray);
253                     explicitLac.rewind();
254                 } else {
255                     throw new IOException("ExplicitLacBufLength " + explicitLacBufLength
256                                           + " is invalid while reading header for " + lf);
257                 }
258             }
259
260             needFlushHeader = false;
261         } else {
262             throw new IOException("Ledger index file " + lf + " does not exist");
263         }
264     }
265
266     public synchronized boolean isDeleted() {
267         return deleted;
268     }

```

Figura 23: Pit coverage

```

233         if (length < 0) {
234
235         1. readHeader : changed conditional boundary → SURVIVED
236         2. readHeader : negated conditional → KILLED
237

```

Figura 24: Pit coverage

Records

Records use the type name "record" and support the following attributes:

- *name*: a JSON string providing the name of the record (required).
- *namespace*, a JSON string that qualifies the name (optional);
- *doc*: a JSON string providing documentation to the user of this schema (optional).
- *aliases*: a JSON array of strings, providing alternate names for this record (optional).
- *fields*: a JSON array, listing fields (required). Each field is a JSON object with the following attributes:
 - *name*: a JSON string providing the name of the field (required), and
 - *doc*: a JSON string describing this field for users (optional).
 - *type*: a [schema](#), as defined above
 - *default*: A default value for this field, only used when reading instances that lack the field for schema evolution purposes. The presence of a default value does not make the field optional at encoding time. Permitted values depend on the field's schema type, according to the table below. Default values for union fields correspond to the first schema in the union. Default values for bytes and fixed fields are JSON strings, where Unicode code points 0-255 are mapped to unsigned 8-bit byte values 0-255. Avro encodes a field even if its value is equal to its default.

Figura 25: Documentazione Record

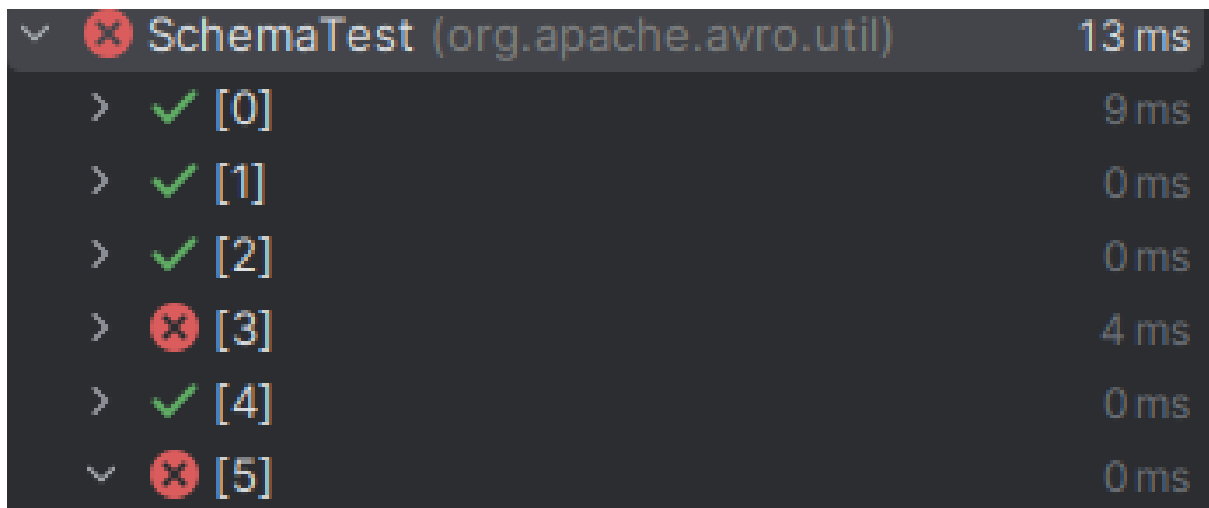


Figura 26: Fallimento del 4 e 6 test

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Schema	<div><div></div></div>	12%	<div><div></div></div>	4%	212	226	363	412	60	72	0	1

Figura 27: Jacoco coverage createRecord

Schema.RecordSchema	<div><div></div></div>	17%	<div><div></div></div>	9%	35	38	84	102	9	11	0	1
---------------------	------------------------	-----	------------------------	----	----	----	----	-----	---	----	---	---

Figura 28: Jacoco coverage Record Schema

Schema.RecordSchema

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
Schema.RecordSchema(Schema.Name, String, boolean, List)	<div><div></div></div>	100%	<div><div></div></div>	n/a	0	1	0	4	0	1
setFields(List)	<div><div></div></div>	59%	<div><div></div></div>	62%	3	5	4	18	0	1

Figura 29: Jacoco coverage Costruttore Record Schema e setField

```

943.     @Override
944.     public void setFields(List<Field> fields) {
945.         if (this.fields != null) {
946.             throw new AvroRuntimeException("Fields are already set");
947.         }
948.         int i = 0;
949.         fieldMap = new HashMap<>(Math.multiplyExact(2, fields.size()));
950.         LockableArrayList<Field> ff = new LockableArrayList<>(fields.size());
951.         for (Field f : fields) {
952.             if (f.position != -1) {
953.                 throw new AvroRuntimeException("Field already used: " + f);
954.             }
955.             f.position = i++;
956.             final Field existingField = fieldMap.put(f.name(), f);
957.             if (existingField != null) {
958.                 throw new AvroRuntimeException(
959.                     String.format("Duplicate field %s in record %s: %s and %s.", f.name(), name, f, existingField));
960.             }
961.             ff.add(f);
962.         }
963.         this.fields = ff.lock();
964.         this.hashCode = NO_HASHCODE;
965.     }
966.

```

Figura 30: Jacoco copertura setField

Schema.RecordSchema





Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed Cxty	Missed Lines	Missed Methods
Schema.RecordSchema(Schema.Name, String, boolean, List)		100%		n/a	0	4	1
setFields(List)		70%		75%	2	18	1

Figura 31: Jacoco coverage dopo incremento

```

-<method name="setFields" desc="(Ljava/util/List;)V">
  <du var="this" def="945" use="945" target="946" covered="0"/>
  <du var="this" def="945" use="945" target="948" covered="1"/>
  <du var="this" def="945" use="949" covered="1"/>
  <du var="this" def="945" use="963" covered="1"/>
  <du var="this" def="945" use="964" covered="1"/>
  <du var="this" def="945" use="956" covered="1"/>
  <du var="this" def="945" use="959" covered="0"/>
  <du var="fields" def="945" use="949" covered="1"/>
  <du var="fields" def="945" use="950" covered="1"/>
  <du var="fields" def="945" use="951" covered="1"/>
  <du var="this.fields" def="945" use="945" target="946" covered="0"/>
  <du var="this.fields" def="945" use="945" target="948" covered="1"/>
  <du var="this.name" def="945" use="959" covered="0"/>
  <du var="i" def="948" use="955" covered="1"/>
  <du var="this.fieldMap" def="949" use="956" covered="1"/>
  <du var="ff" def="950" use="963" covered="1"/>
  <du var="ff" def="950" use="961" covered="1"/>
  <du var="f" def="951" use="952" target="953" covered="1"/>
  <du var="f" def="951" use="952" target="955" covered="1"/>
  <du var="f" def="951" use="955" covered="1"/>
  <du var="f" def="951" use="956" covered="1"/>
  <du var="f" def="951" use="961" covered="1"/>
  <du var="f" def="951" use="959" covered="0"/>
  <du var="f" def="951" use="959" covered="0"/>
  <du var="f" def="951" use="953" covered="1"/>
  <du var="i" def="955" use="955" covered="1"/>
  <du var="existingField" def="956" use="957" target="958" covered="0"/>
  <du var="existingField" def="956" use="957" target="961" covered="1"/>
  <du var="existingField" def="956" use="959" covered="0"/>
  <counter type="DU" missed="8" covered="24"/>
  <counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 32: Badua coverage createRecord

```

<method name="setFields" desc="(Ljava/util/List;)V">
  <du var="this" def="945" use="945" target="946" covered="0"/>
  <du var="this" def="945" use="945" target="948" covered="1"/>
  <du var="this" def="945" use="949" covered="1"/>
  <du var="this" def="945" use="963" covered="1"/>
  <du var="this" def="945" use="964" covered="1"/>
  <du var="this" def="945" use="956" covered="1"/>
  <du var="this" def="945" use="959" covered="1"/>
  <du var="fields" def="945" use="949" covered="1"/>
  <du var="fields" def="945" use="950" covered="1"/>
  <du var="fields" def="945" use="951" covered="1"/>
  <du var="this.fields" def="945" use="945" target="946" covered="0"/>
  <du var="this.fields" def="945" use="945" target="948" covered="1"/>
  <du var="this.name" def="945" use="959" covered="1"/>
  <du var="i" def="948" use="955" covered="1"/>
  <du var="this.fieldMap" def="949" use="956" covered="1"/>
  <du var="ff" def="950" use="963" covered="1"/>
  <du var="ff" def="950" use="961" covered="1"/>
  <du var="f" def="951" use="952" target="953" covered="1"/>
  <du var="f" def="951" use="952" target="955" covered="1"/>
  <du var="f" def="951" use="955" covered="1"/>
  <du var="f" def="951" use="956" covered="1"/>
  <du var="f" def="951" use="961" covered="1"/>
  <du var="f" def="951" use="959" covered="1"/>
  <du var="f" def="951" use="959" covered="1"/>
  <du var="f" def="951" use="953" covered="1"/>
  <du var="i" def="955" use="955" covered="1"/>
  <du var="existingField" def="956" use="957" target="958" covered="1"/>
  <du var="existingField" def="956" use="957" target="961" covered="1"/>
  <du var="existingField" def="956" use="959" covered="1"/>
  <counter type="DU" missed="2" covered="30"/>
  <counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 33: Badua coverage createRecord




Name	Line Coverage	Mutation Coverage	Test Strength
Schema.java	15%  132/894	4%  23/570	51%  23/45

Figura 34: PitCoverage

```

222  /** Create a named record schema with fields already set. */
223  public static Schema createRecord(String name, String doc, String namespace, boolean isError, List<Field> fields) {
224      return new RecordSchema(new Name(name, namespace), doc, isError, fields);
225  }

```

Figura 35: Mutazioni rilevate da Pitest

```

913  public RecordSchema(Name name, String doc, boolean isError, List<Field> fields) {
914      super(Type.RECORD, name, doc);
915      this.isError = isError;
916      setFields(fields);
917  }

```

Figura 36: Mutazioni rilevate da Pitest


```

943     @Override
944     public void setFields(List<Field> fields) {
945         if (this.fields != null) {
946             throw new AvroRuntimeException("Fields are already set");
947         }
948         int i = 0;
949         fieldMap = new HashMap<>(Math.multiplyExact(2, fields.size()));
950         LockableArrayList<Field> ff = new LockableArrayList<>(fields.size());
951         for (Field f : fields) {
952             if (f.position != -1) {
953                 throw new AvroRuntimeException("Field already used: " + f);
954             }
955             f.position = i++;
956             final Field existingField = fieldMap.put(f.name(), f);
957             if (existingField != null) {
958                 throw new AvroRuntimeException(
959                     String.format("Duplicate field %s in record %s: %s and %s.", f.name(), name, f, existingField));
960             }
961             ff.add(f);
962         }
963         this.fields = ff.lock();
964         this.hashCode = NO_HASHCODE;
965     }
966

```

Figura 37: Mutazioni rilevate da Pitest

```

222  /** Create a named record schema with fields already set. */
223  1. createRecord : replaced return value with null for org/apache
224  /avro/Schema::createRecord → SURVIVED
225  }
226

```

Figura 38: Mutazione in considerazione

```

221
222  /** Create a named record schema with fields already set. */
223  public static Schema createRecord(String name, String doc, String namespace, boolean isError, List<Field> fields) {
224  1. return new RecordSchema(new Name(name, namespace), doc, isError, fields);
225  }

```

Figura 39: Mutazione in considerazione coperta

SchemaCompatibility

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
checkReaderWriterCompatibility(Schema, Schema)		78%		66%	1	3	1	9	0	1
static [...]		81%		50%	1	2	0	2	0	1
lookupWriterField(Schema, Schema.Field)		0%		0%	8	8	14	14	1	1
schemaNameEquals(Schema, Schema)		0%		0%	2	2	3	3	1	1
objectsEqual(Object, Object)		0%		n/a	1	1	1	1	1	1
asList(Deque)		100%		n/a	0	1	0	3	0	1
Total	105 of 167	37%	17 of 20	15%	13	17	19	32	3	6

Figura 40: Coverage Jacoco

```

60.  public static SchemaPairCompatibility checkReaderWriterCompatibility(final Schema reader, final Schema writer) {
61.  final SchemaCompatibilityResult compatibility = new ReaderWriterCompatibilityChecker().getCompatibility(reader,
62.  writer);
63.
64.  final String message;
65.  switch (compatibility.getCompatibility()) {
66.  case INCOMPATIBLE: {
67.  message = String.format(
68.  "Data encoded using writer schema:%n%s%n" + "will or may fail to decode using reader schema:%n%s%n",
69.  writer.toString(true), reader.toString(true));
70.  break;
71.  }
72.  case COMPATIBLE: {
73.  message = READER_WRITER_COMPATIBLE_MESSAGE;
74.  break;
75.  }
76.  default:
77.  throw new AvroRuntimeException("Unknown compatibility: " + compatibility);
78.  }
79.
80.  return new SchemaPairCompatibility(compatibility, reader, writer, message);
81.  }
82.

```

Figura 41: Coverage Jacoco

```

-<class name="org/apache/avro/SchemaCompatibility">
-<method name="checkReaderWriterCompatibility" desc="(Lorg/apache/avro/Schema;Lorg/apache/avro/Schema;)Lorg/apache/avro/SchemaCompatibility$SchemaPairCompatibility;">
  <du var="reader" def="61" use="80" covered="1"/>
  <du var="reader" def="61" use="69" covered="1"/>
  <du var="writer" def="61" use="80" covered="1"/>
  <du var="writer" def="61" use="69" covered="1"/>
  <du var="$SwitchMap$org$apache$avro$SchemaCompatibility$SchemaCompatibilityType" def="61" use="65" target="77" covered="0"/>
  <du var="$SwitchMap$org$apache$avro$SchemaCompatibility$SchemaCompatibilityType" def="61" use="65" target="67" covered="1"/>
  <du var="$SwitchMap$org$apache$avro$SchemaCompatibility$SchemaCompatibilityType" def="61" use="65" target="73" covered="1"/>
  <du var="compatibility" def="61" use="65" target="77" covered="0"/>
  <du var="compatibility" def="61" use="65" target="67" covered="1"/>
  <du var="compatibility" def="61" use="65" target="73" covered="1"/>
  <du var="compatibility" def="61" use="80" covered="1"/>
  <du var="compatibility" def="61" use="77" covered="0"/>
  <du var="message" def="67" use="80" covered="1"/>
  <du var="message" def="73" use="80" covered="1"/>
  <counter type="DU" missed="3" covered="11"/>
  <counter type="METHOD" missed="0" covered="1"/>
</method>

```

Figura 42: Coverage Badua

```

50 public static SchemaPairCompatibility checkReaderWriterCompatibility(final Schema reader, final Schema writer) {
51     final SchemaCompatibilityResult compatibility = new ReaderWriterCompatibilityChecker().getCompatibility(reader,
52         writer);
53
54     final String message;
55     switch (compatibility.getCompatibility()) {
56     case INCOMPATIBLE: {
57         message = String.format(
58             "Data encoded using writer schema:%n%s%n" + "will or may fail to decode using reader schema:%n%s%n",
59             writer.toString(true), reader.toString(true));
60         break;
61     }
62     case COMPATIBLE: {
63         message = READER_WRITER_COMPATIBLE_MESSAGE;
64         break;
65     }
66     default:
67         throw new AvroRuntimeException("Unknown compatibility: " + compatibility);
68     }
69
70     return new SchemaPairCompatibility(compatibility, reader, writer, message);
71 }
72
73 // -----
74 /**

```

Figura 43: Coverage Pit

```

8 public class SchemaSchemaResolverIT {
9
10     @Test
11     public void test() {
12         String name = "test";
13         Schema s = SchemaResolver.unresolvedSchema(name);
14         assertEquals(name, s.getProp( name: "org.apache.avro.compiler.idl.unresolved.name"));
15     }
16 }

```

Figura 44: Codice Integration Test

```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <failureSummary xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation="https://maven.apache.org/surefire/maven-surefire-plugin/xsd/failure-summary.xsd" result="null" timeout="false">
3     <completed>0</completed>
4     <errors>0</errors>
5     <failures>0</failures>
6     <skipped>0</skipped>
7     <failureMessage xsi:nil="true" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"/>
8 </failureSummary>

```

Figura 45: Report Integration Test