Ingegneria del Software 2 – ISW2

Modulo Software Testing

Di Simone Mesiano Laureani

0278325

email: simesi@hotmail.it

Introduzione

Il seguente report è stato redatto prendendo in considerazione due progetti open-source sviluppati dall'Apache Software Foundation e assegnati sulla base dell'algoritmo presentato durante il corso. I due progetti stabiliti sono stati quindi Bookkeeper [1] e OpenJPA [2]. L'obiettivo di questo report è quello di mostrare la metodologia seguita e i risultati ottenuti dall'attività di testing su due classi java per ogni progetto. Tali classi sono state scelte sulla base dei risultati ottenuti dalle attività svolte durante e successivamente al modulo del corso di ISW2 con il Prof. Falessi e specificatamente si è tenuto conto della propensione di alcune delle classi ad essere classificate come "Buggy" sull'analisi dei ticket ottenuti da Jira [3]. L'ambiente di lavoro per i due progetti è stato configurato partendo da un fork su Github delle rispettive repositories [4] [5]. Si è utilizzato Travis CI [6] per il building Maven in remoto e SonarCloud [7] per l'analisi dei test effettuati e delle varie coverage ottenute. Infine, è stato utilizzato il framework di JUnit [8] per l'effettiva implementazione dei casi di test, JaCoCo [9] per la generazione dei report dei risultati dei test e PIT [10] per l'applicazione delle mutazioni alle classi in esame.

Problematiche affrontate

Il setup dei progetti ha richiesto una quantità di tempo superiore a quella attesa a causa di vari fattori come ad esempio la cancellazione iniziale di tutti i test nativi provocava l'emergere di errori nella build di entrambi i progetti e solo in seguito si è compreso di dover modificare di conseguenza anche i relativi file *pom.xml*. In aggiunta a ciò si sono presentati dei problemi riguardanti il collegamento tra SonarCloud e i progetti in esame in quanto il file *sonar-project.properties* proposto in [11] non è sembrato sortire l'effetto desiderato e solo in seguito si è risolto inserendo direttamente all'interno del *pom.xml* principale le informazioni di collegamento necessarie al proprio account di SonarCloud. Inoltre, durante la scrittura dei casi di test, utilizzando l'IDE Eclipse [12] in ambiente Windows, è sorto un problema riguardante la formattazione del testo che alla build dei progetti faceva risultare un errore nella parte di *checksum* dato dalla scrittura in formato DOS su UNIX format e di conseguenza risultavano vari messaggi di Warning con la dicitura "*LF will be replaced by CRLF in file […]"*. Tale problematica è stata superata aggiungendo linee vuote alla fine di ogni file scritto. In aggiunta a ciò è sorto un problema con l'esecuzione del framework PIT in ambiente Windows, il quale faceva scaturire l'errore *"could not find org.pitest.coverage.execute.CoverageMinion"* e dopodiché si passava in stato di freeze per un tempo indefinito. Tale problema non è stato ancora risolto in ambiente Windows nonostante la visione di tutta la (poca) documentazione inerente a questo specifico errore e quindi si è provveduto ad eseguire tale operazione di Mutation Testing in un ambiente Linux offerto da una macchina virtuale tramite l'applicazione di virtualizzazione Virtualbox [13]. Infine molte classi che inizialmente sono state ritenute idonee per effettuarvi attività di testing, si sono dovute abbandonare perché di tipo *final* o *private* e quindi non mockabili, oppure non implementavano getters e/o i metodi restituivano *void* quindi vi era un'impossibilità di ottenere lo "stato" modificato dei singoli oggetti in seguito alle operazioni effettuate. Ad esempio si voleva analizzare la classe ApplicationIds presente in org/apache/openjpa/util/ ma non è stato possibile in quanto i metodi di tale classe prendevano come input un oggetto complesso di tipo `ClassMetaData` con costruttore *protected.* Un altro esempio di classe scartata, ma questa volta in Bookkeeper, è stata BookieSocketAddress la quale è stata interessata da fix di improvement (BOOKKEEPER 1008 e 612) nella versione 13 e da fix di bug. L'analisi di tale classe è stata abbandonata perchè non è possibile mockare metodi forniti da java.net ed per di più anche *final.*

OpenJPA

OpenJPA è l'implementazione di Apache della specifica Java Persistence API (JPA) pensata per semplificare il salvataggio di oggetti nei database. Tale API è un framework per il linguaggio di programmazione Java che si occupa della gestione della persistenza dei dati di un DBMS relazionale nelle applicazioni che usano le piattaforme Java Platform, Standard Edition e Java Enterprise Edition. La maggior parte dei programmi utilizza dati persistenti per il corretto funzionamento delle operazioni a loro affidate. Tali dati devono essere salvati in supporti di memorizzazione per persistere anche dopo l'esecuzione del programma. JPA offre delle specifiche per il meccanismo di persistenza delle informazioni: tale meccanismo permette di salvare i dati delle classi all'interno di un database relazionale in modo trasparente allo sviluppatore. Grazie a questa API lo sviluppatore non è più direttamente coinvolto nella gestione del salvataggio dei dati, diminuendo lo sforzo necessario per lo sviluppo delle applicazioni. Le classi che sono state selezionate per le attività di testing sono*: Specification.java* e *ArrayStateImage.java*. La prima classe è stata scelta in quanto segnalata come "buggy" nella versione 13 e 14 (con id di versione M2 e M3) dal report ottenuto dalla parte di progetto richiesta dal prof. Falessi. Nello studio dell'evoluzione nel tempo di tale classe, riportato nella Tabella 1, si è evidenziato come essa sia stata introdotta nella versione 11 del progetto e in seguito solo leggermente modificata. Possiamo quindi affermare che molto probabilmente si è di fronte a bugs 'dormienti' nel codice e che quindi solo in seguito vengono scoperti e corretti. I primi test sono stati condotti sul costruttore di tale classe:

```
public Specification(String fullName)
```

Tale costruttore banalmente divideva la stringa passata in input e ne verificava la consistenza in base a un formato specifico. La stringa poteva contenere uno o più parole/numeri ad indicare i campi nome e versione per il 'major' e il 'minor'. Tale costruttore poteva lanciare una eccezione specifica in caso di formato sbagliato. Seguendo i concetti del category partition studiato durante il corso, si sono individuate le partizioni considerando almeno un insieme di stringhe tutte valide e un insieme di stringhe tutte non valide. Un altro metodo preso in esame è il seguente:

```
public boolean equals(Object other)
```

Tale metodo è un override di quello proposto da java nativamente e si occupa di verificare l'equivalenza dello 'stato' dell'oggetto passato con quello da cui è stato lanciato il suddetto metodo. Durante lo studio di tale metodo di è scoperta l'effettiva presenza di un bug nell'istruzione di ritorno del metodo in esame presente tutt'ora. In Figura 1 è presente uno snippet di codice che lo mostra. Si può notare infatti come questo metodo ritorni *true* anche se i nomi dei due oggetti e/o i minors siano in realtà diversi tra loro (infatti al metodo occorre solo che il major sia uguale per giudicare i due oggetti uguali). Tale errore deve essere figlio di una semplice distrazione da parte di uno dei quattro committer che hanno lavorato alla scrittura di tale classe. Come input per i test si sono identificati i domini di partizione e si è sviluppato un caso di test per ognuno di essi (oggetto valido, invalido e null). Gli altri metodi della classe non sono stati presi in esame in quanto consistenti di una sola riga di codice come implementazione e quindi ritenuti troppo facili. In *Figura 2* è possibile vedere il report generato da JaCoCo dopo l'implementazione dei partition test. Invece in *Figura 3* è possibile vedere i risultati ottenuti da SonarCloud. Dopo la visione di tali risultati si è proceduto ad aggiungere l'insieme dei test per migliorare le metriche di adeguatezza e si sono ottenuti i risultati visibili in *Figura 4 e 5.*

La seconda classe presa in esame è stata scelta in contrapposizione alla prima (avendo un Age elevata): essa infatti è stata creata già dalla prima versione del progetto OpenJPA e quindi occupa un ruolo fondamentale di supporto all'archiviazione di dati. Il primo metodo analizzato è:

```
static Object[] newImage(int numFields)
```

Tale metodo crea "un'immagine" che consiste nel popolamento di un array di Objects con numFileds valori e come ultima entry di tale array venivano inseriti dei bit ad indicare i campi inseriti. Come test iniziali si sono passati {-1,0,1} perché di interesse per il category partition. Un altro metodo analizzato è:

```
boolean isImage(Object obj)
```

Tale metodo ritorna *true* se l'oggetto passatogli contiene dei valori nello specifico formato BitSet oltre a non essere di taglia nulla. Per l'implementazione di test su tale metodo ho utilizzato il metodo newImage() precedente (non mockato) come input a isImage() in modo da creare un oggetto valido. Un altro metodo analizzato è:

```
static BitSet getLoaded(Object[] state)
```

Questo metodo ritorna i BitSet associati "all'immagine" passata in oggetto. Nell'identificare le classi di partizione non si è potuto inserire un array tale da superare la lunghezza massima consentita in quanto non vi era un limite ad essa. Un altro metodo preso in esame è:

```
void setLoaded(Object[] state, BitSet loaded)
```

Il metodo in esame modificava semplicemente lo stato dei BitSet presenti nell'array *state* con quelli passati come parametro *loaded*. Si sono sviluppati vari casi di test di tipo monodimensionale includendo quindi istanze valide, invalide e nulle per entrambi i parametri. Un altro metodo esaminato è stato:

```
Object[] clone(Object[] state)
```

Con tale metodo si clona lo stato preso come input creando un nuovo array di Objects e settando all'interno dei valori di stato sulla base dell'array ricevuto. Come in precedenza riferito per altri metodi, si è iniziato a sviluppare casi di test sulla base del Category Partition e quindi si sono scelti come input un'array vuoto, uno non vuoto valido e uno non vuoto invalido. Infine l'ultimo metodo analizzato di questa classe (e anche il più lungo) è:

```
static boolean sameVersion(Object[] state1, Object[] state2)
```

Tale metodo ritorna vero se, come dice il nome, le immagini data sono equivalenti tra loro. Da notare che se entrambe le immagini sono nulle allora il metodo ritorna comunque *true* in quanto una versione nulla indica

che non ci sono campi nella versione e che quindi non esiste nulla da comparare. Quindi tale metodo permette l'inserimento (senza il raise di eccezioni) di parametri *null*. La comparazione è fatta per ogni campo presente nello *state1* finchè essi non finiscono o la comparazione ritorna *false.* In *figura 6* è possibile vedere il report generato da JaCoCo. Nel miglioramento della coverage dei metodi sopracitati, si è scoperto che non è possibile aumentare la branch coverage del metodo IsImage() perchè se la prima condizione nel return è falsa, allora la seconda non può mai essere vera. A tale proposito si veda lo snippet di codice mostrato in *figura 8*. Dopo aver raffinato ed aumentato i test case, si è raggiunti al risultato mostrato nelle *Figure 7 e 9*. In seguito, si sono applicate le mutazioni tramite il framework PIT e i risultati sono visibili in *Figura 10.* I risultati ottenuti sono considerati abbastanza ottimali per i casi di test delle due classi prese in esame.

Bookkeeper

Bookkeeper è un servizio di storage scalabile, tollerante ai guasti e a bassa latenza, ottimizzato per carichi di lavoro real-time. Tale progetto fino ad ora è stato sviluppato in 14 versioni differenti, di cui l'ultima è stata sviluppato nel 2017. Le classi prese in esame sono: Value e MathUtils. In *Figura 12* è presente un'estratto del report ottenuto dal progetto richiesto dal prof. Falessi che riguarda la classe Value. Si può notare come la classe in oggetto presenti un alto valore di Loc_Touched nella versione 3 del progetto. Codesta classe è stata modificata in seguito ai ticket 561 e 940 che indicavano la presenza di bug che quindi interessavano la versione 4.3 e 4.5 del progetto Bookkeeper. La classe è stata modificata anche in seguito ai ticket 204 e 1010 per fattori di improvements. Tra i metodi della classe Value considerati si è analizzato il metodo:

```
Value project(Set<String> fields)
```

Tale metodo fa una proiezione dei valori ottenuti in input creando un oggetto complesso di tipo Value settandone opportunamente i campi. Si è cominciato a fornire come input ai casi di test un set di stringhe valido, uno vuoto e uno invalido. In questi ultimi due casi il suddetto metodo ritorna una copia identica dell'oggetto Value. Altrimenti come sopra citato, si effettua una proiezione dei campi presenti. Tale operazione richiama l'algebra relazionale SQL e quindi l'operazione di "Select". Un altro metodo preso in esame è:

```
Value merge(Value other)
```

Questo metodo prende lo stato dell'oggetto *other* e ne scandisce i vari valori. Nel dettaglio, il valore di ogni field è controllato e se è null allora viene cancellata la propria chiave associato. Quindi si fa una sorta di update delle informazioni presenti all'interno dello stato dell'oggetto su cui è chiamato il metodo. I valori vecchi vengono aggiornati rimuovendo le chiavi non presenti nell'oggetto *other.* Infine l'ultimo metodo analizzato è:

```
@Override
public String toString()
```

Tale metodo è, come ne suggerisce l'annotazione, un override dell'implementazione basica fornita da Java. Il metodo si avvale di uno StringBuilder per la concatenazione della coppia chiave-valore di ogni campo rappresentante lo stato dell'oggetto su cui è chiamato tale metodo. In caso di presenza di chiave nulla, viene introdotta una stringa "NULL", mentre in caso di valore mancante si introduce una stringa "NONE". In F*igura 13* è mostrato il report ottenuto da JaCoCo dopo i Category Partition test. In *Figura 14* e in *Figura 15* è mostrata la copertura identificata dall'IDE Eclipse sempre dopo i Category Partition test. Vedendo tali risultati si è provveduto a raffinare i casi di test ed ad aumentarli di numero. In seguito a questa attività, si sono ottenuti i risultati visibili in *Figura 16* ottenuti grazie a JaCoCo. Come ultimo procedimento si sono applicate delle mutazioni con il framework PIT e i risultati sono visibili in *Figura 17.*

L’altra classe presa in esame è MathUtils, presente nel modulo bookkeeper-common. Tale classe come ne suggerisce il nome, offre delle funzioni matematiche che vanno oltre quelle standard offerte da java. Il primo metodo preso in esame è:

```
int signSafeMod(long dividend, int divisor)
```

Esso implementa una versione della funzione di modulo però reinterpretata in modo da rimanere safe in caso di valori negativi passati come input. Come valori per gli input dei test iniziali si sono passati un valore valido e uno invalido (nel senso di negativo) per entrambi i parametri. L’altro metodo considerato è:

```
static int findNextPositivePowerOfTwo(final int value)
```

Il metodo sopra mostrato ritorna il più piccolo numero maggiore o uguale a quello dato in input tale che sia anche una potenza di due. Tale metodo fa affidamento all’implementazione di numberOfLeadingZeros() offerta da java. Gli altri metodi nella classe non sono stati presi in considerazione perché consistenti di una sola riga di codice come implementazione del comportamento. In *figura 18* sono presenti i risultati ottenuti da JaCoCo dopo la formulazione dei Category Partition test. Invece in *Figura 19* è presente il report fornito da Eclipse e in Figura 20 è riproposto lo stesso riepilogo ma dopo dei miglioramenti. Inoltre si può osservare in *Figura 21* il rapporto fornito da JacoCo dopo i vari raffinamenti del caso. Infine si è provveduto ad eseguire PIT per la configurazione di mutazioni e il lancio di mutanti. I risultati di tale attività è mostrata in *Figura 22*.

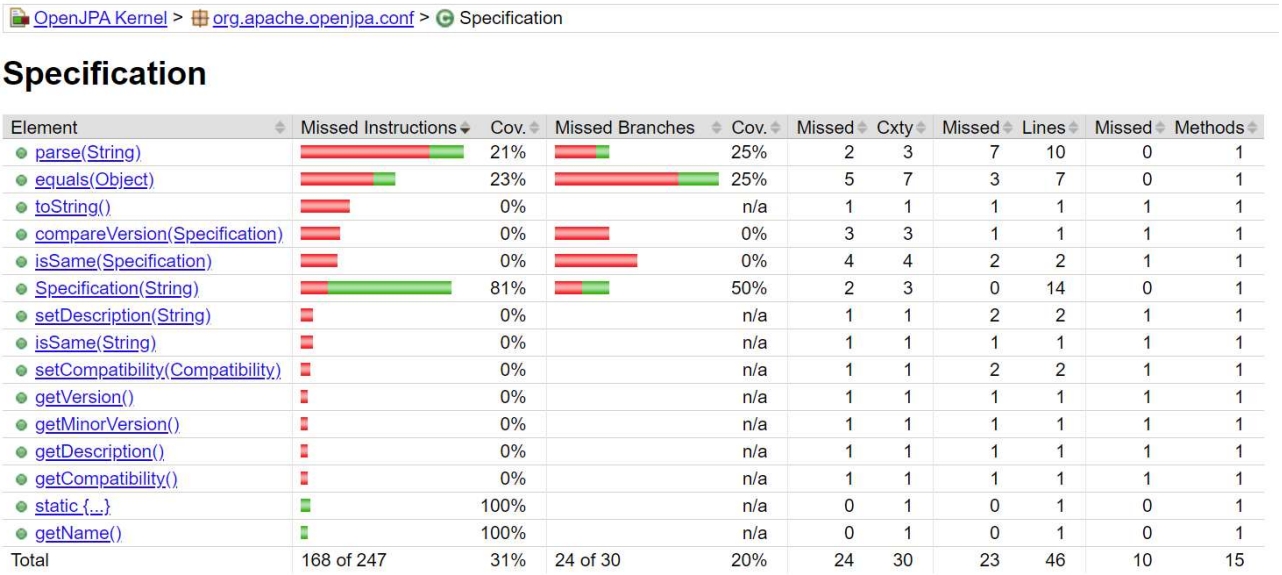
Risultati ottenuti

Version	File Name	Size(LOC)	LOC_Touched	NR	NAuth	LOC_Added	MAX_LOC_Added	AVG_LOC_Added	Churn	MAX_Churn	AVG_Churn	Buggy
11	openjpa-kernel/src/main/java/org/apache/openjpa/conf/Specification.java	147	385	2	1	208	208	0	86	147	43	NO
12	openjpa-kernel/src/main/java/org/apache/openjpa/conf/Specification.java	147	294	3	2	0	147	0	86	147	28	NO
13	openjpa-kernel/src/main/java/org/apache/openjpa/conf/Specification.java	147	0	3	2	0	0	0	86	147	28	YES
14	openjpa-kernel/src/main/java/org/apache/openjpa/conf/Specification.java	165	18	4	3	18	18	0	104	147	26	YES
15	openjpa-kernel/src/main/java/org/apache/openjpa/conf/Specification.java	165	0	4	3	0	0	0	104	147	26	NO
16	openjpa-kernel/src/main/java/org/apache/openjpa/conf/Specification.java	165	0	4	3	0	0	0	104	147	26	NO
17	openjpa-kernel/src/main/java/org/apache/openjpa/conf/Specification.java	165	0	4	3	0	0	0	104	147	26	NO

Tabella 1

```
87      /**
88       * Affirms if the given argument is equal to this receiver.
89       */
90      @Override
91      public boolean equals(Object other) {
92          if (this == other)
93              return true;
94          if (other == null || !this.getClass().isInstance(other))
95              return false;
96          Specification that = (Specification)other;
97          return Objects.equals(_name, this._name) && _major == that._major
98              && Objects.equals(_minor, this._minor);
99      }
```

Figure 1



OpenJPA Kernel > org.apache.openjpa.conf > Specification

Specification

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
parse(String)		21%		25%	2	3	7	10	0	1
equals(Object)		23%		25%	5	7	3	7	0	1
toString()		0%		n/a	1	1	1	1	1	1
compareVersion(Specification)		0%		0%	3	3	1	1	1	1
isSame(Specification)		0%		0%	4	4	2	2	1	1
Specification(String)		81%		50%	2	3	0	14	0	1
setDescription(String)		0%		n/a	1	1	2	2	1	1
isSame(String)		0%		n/a	1	1	1	1	1	1
setCompatibility(Compatibility)		0%		n/a	1	1	2	2	1	1
getVersion()		0%		n/a	1	1	1	1	1	1
getMinorVersion()		0%		n/a	1	1	1	1	1	1
getDescription()		0%		n/a	1	1	1	1	1	1
getCompatibility()		0%		n/a	1	1	1	1	1	1
static {...}		100%		n/a	0	1	0	1	0	1
getName()		100%		n/a	0	1	0	1	0	1
Total	168 of 247	31%	24 of 30	20%	24	30	23	46	10	15

Figure 2 After Partition test

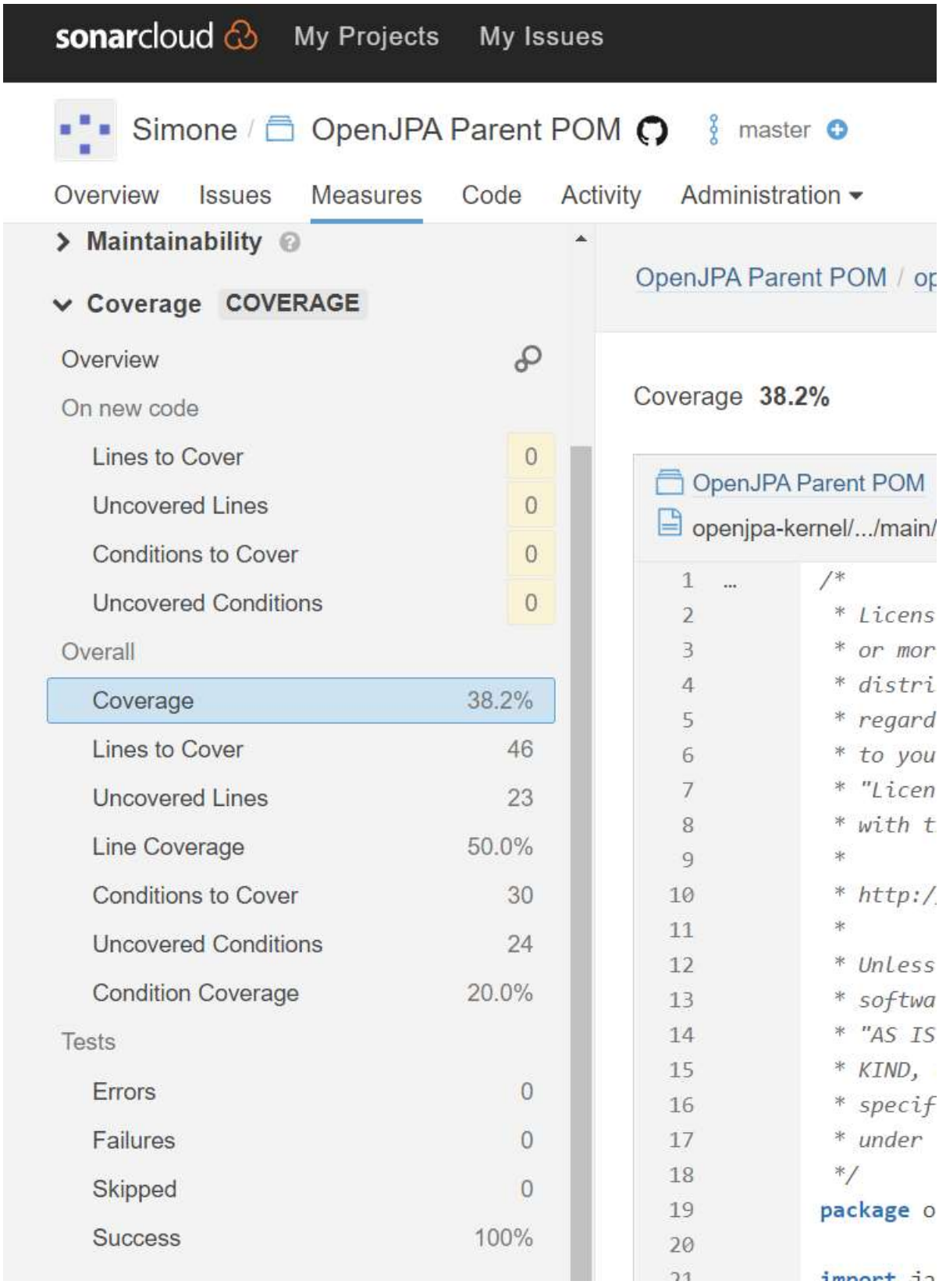


Figure 3 After Partition Test (Specification.java)

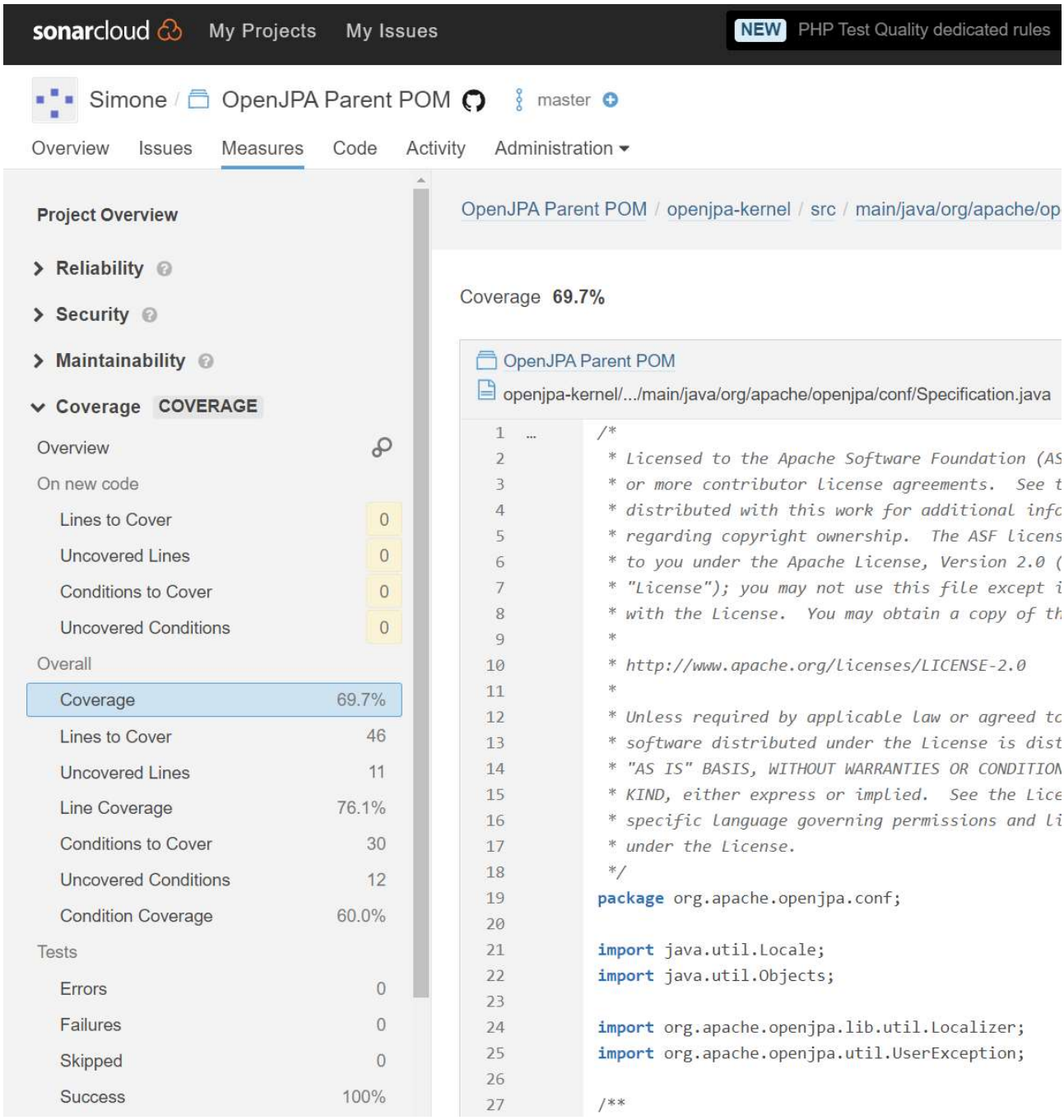


Figure 4 After improvements

OpenJPA Kernel > org.apache.openjpa.conf > Specification

Specification

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
toString()		0%		n/a	1	1	1	1	1	1
compareVersion(Specification)		0%		0%	3	3	1	1	1	1
isSame(Specification)		0%		0%	4	4	2	2	1	1
setDescription(String)		0%		n/a	1	1	2	2	1	1
isSame(String)		0%		n/a	1	1	1	1	1	1
setCompatibility(Compatibility)		0%		n/a	1	1	2	2	1	1
getDescription()		0%		n/a	1	1	1	1	1	1
getCompatibility()		0%		n/a	1	1	1	1	1	1
parse(String)		100%		100%	0	3	0	10	0	1
Specification(String)		100%		100%	0	3	0	14	0	1
equals(Object)		100%		83%	2	7	0	7	0	1
static {...}		100%		n/a	0	1	0	1	0	1
getName()		100%		n/a	0	1	0	1	0	1
getVersion()		100%		n/a	0	1	0	1	0	1
getMinorVersion()		100%		n/a	0	1	0	1	0	1
Total	71 of 247	71%	12 of 30	60%	15	30	11	46	8	15

Figure 5 After improvements

OpenJPA Kernel > org.apache.openjpa.util > ArrayStateImage

ArrayStateImage

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
sameVersion(Object[], Object[])		87%		57%	6	8	3	11	0	1
ArrayStateImage()		0%		n/a	1	1	1	1	1	1
isImage(Object)		95%		66%	2	4	0	4	0	1
clone(Object[])		100%		n/a	0	1	0	5	0	1
newImage(int)		100%		n/a	0	1	0	3	0	1
getLoaded(Object[])		100%		n/a	0	1	0	1	0	1
setLoaded(Object[], BitSet)		100%		n/a	0	1	0	2	0	1
Total	10 of 135	92%	8 of 20	60%	9	17	4	27	1	7

Figure 6 After Partition test

OpenJPA Kernel > org.apache.openjpa.util > ArrayStateImage

ArrayStateImage

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
ArrayStateImage()		0%		n/a	1	1	1	1	1	1
sameVersion(Object[], Object[])		100%		100%	0	8	0	11	0	1
clone(Object[])		100%		n/a	0	1	0	5	0	1
isImage(Object)		100%		83%	1	4	0	4	0	1
newImage(int)		100%		n/a	0	1	0	3	0	1
getLoaded(Object[])		100%		n/a	0	1	0	1	0	1
setLoaded(Object[], BitSet)		100%		n/a	0	1	0	2	0	1
Total	3 of 135	97%	1 of 20	95%	2	17	1	27	1	7

Figure 7 After improvements

```
 * @author Abe White
 */
public class ArrayStateImage {

    /**
     * Create a new state image for the given number of fields.
     */
    public static Object[] newImage(int numFields) {
        Object[] state = new Object[numFields + 1];
        state[numFields] = new BitSet(numFields);
        return state;
    }

    /**
     * Return true if the given version object appears to be an array state
     * image.
     */
    public static boolean isImage(Object obj) {
        if (!(obj instanceof Object[]))
            return false;
        Object[] arr = (Object[]) obj;
        return arr.length > 0 && arr[arr.length - 1] instanceof BitSet;
    }

    /**
     * Get the loaded mask from a state image.
     */
    public static BitSet getLoaded(Object[] state) {
        return (BitSet) state[state.length - 1];
    }

    /**
     * Set the loaded mask into a state image.
     */
    public static void setLoaded(Object[] state, BitSet loaded) {
        state[state.length - 1] = loaded;
    }
```

Figure 8 Uncovered branches (ArrayStateImage)

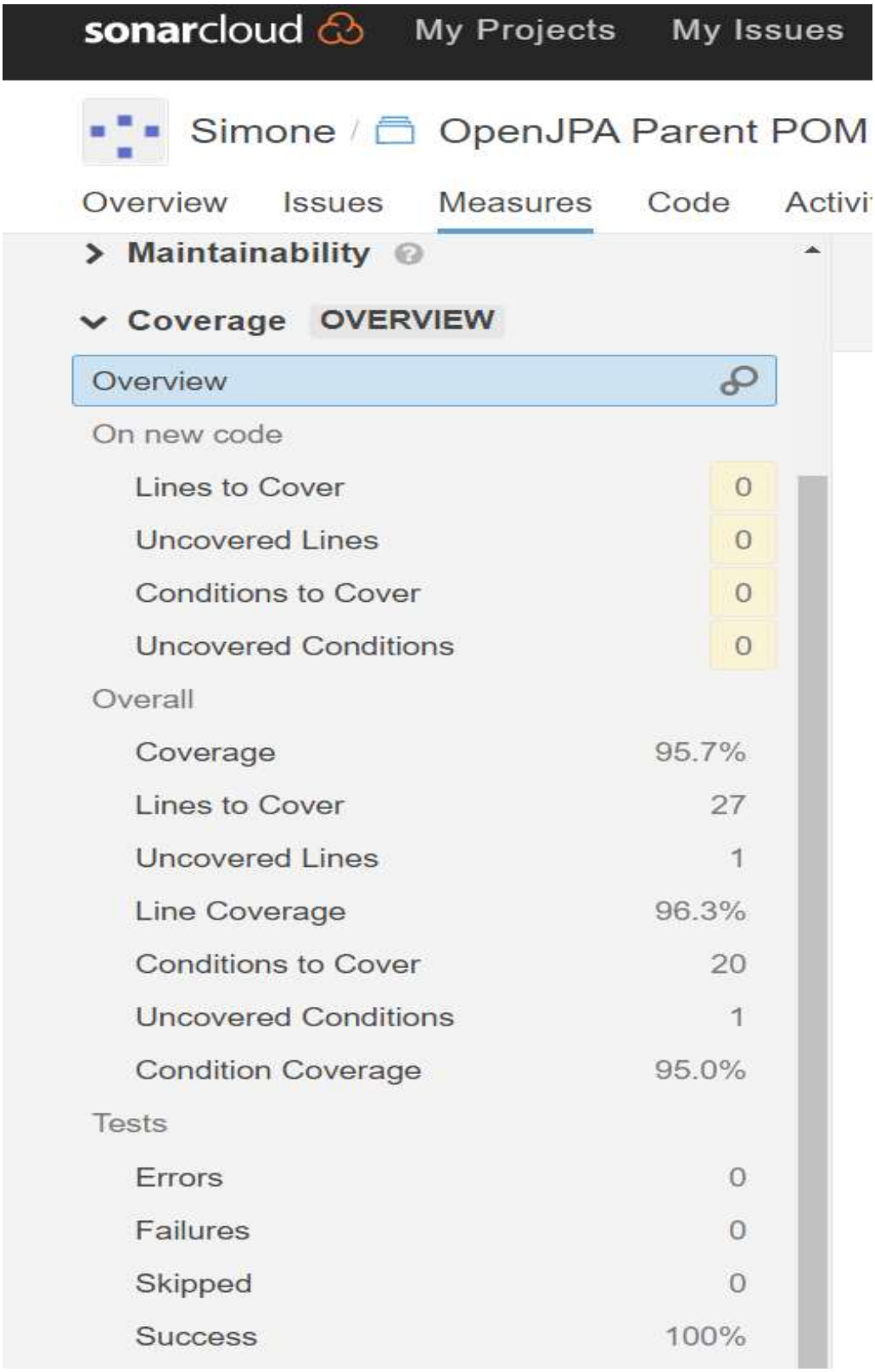


Figure 9 After improvements

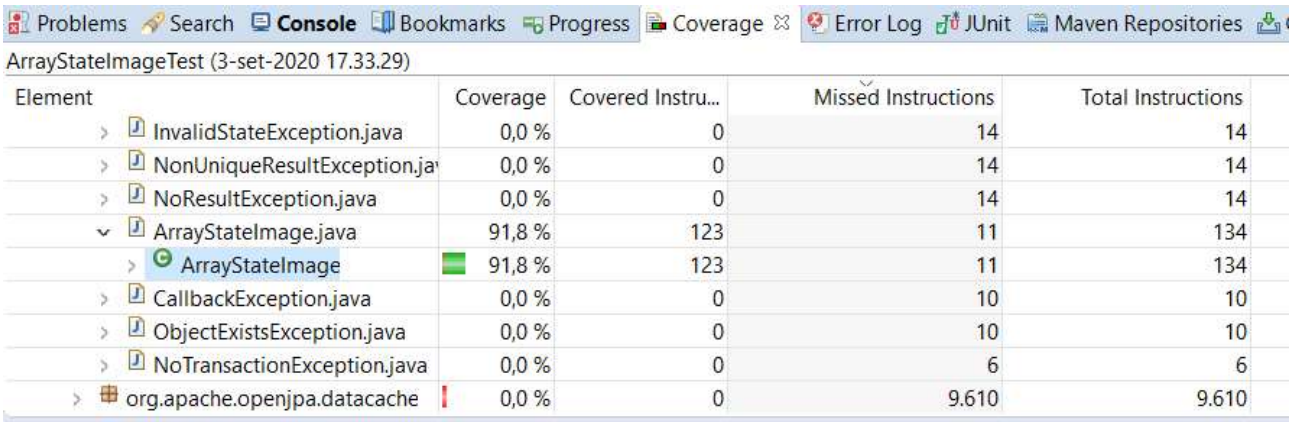


Figure 10 After improvements (ArrayStateImage)

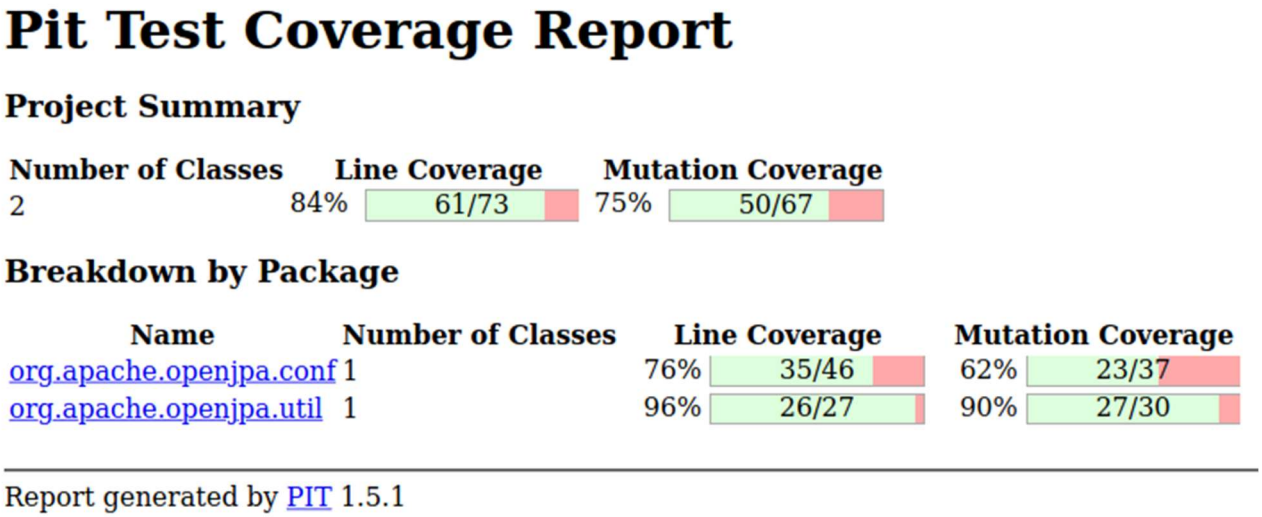
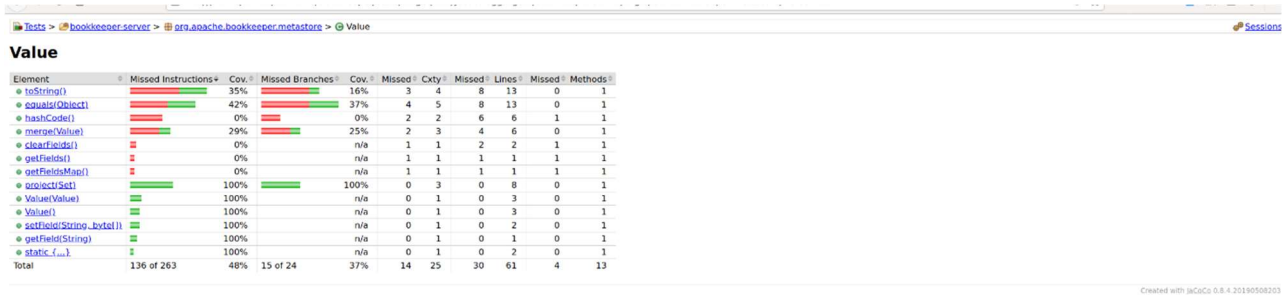


Figure 11 Report PIT (classi Specification e ArrayStateImage)

Vers	File Name	Size(LOC)	LOC_Touched	NR	NAuth	LOC_Added	MAX_LOC_Added	AVG_LOC_Added	Churn	MAX_Churn	AVG_Churn
3	bookkeeper-server/src/main/java/org/apache/bookkeeper/metastore/Value.java	154	154	1	1	154	154	0	154	154	154
4	bookkeeper-server/src/main/java/org/apache/bookkeeper/metastore/Value.java	154	0	1	1	0	0	0	154	154	154
5	bookkeeper-server/src/main/java/org/apache/bookkeeper/metastore/Value.java	156	4	2	2	2	3	0	156	154	78
6	bookkeeper-server/src/main/java/org/apache/bookkeeper/metastore/Value.java	156	0	2	2	0	0	0	156	154	78
7	bookkeeper-server/src/main/java/org/apache/bookkeeper/metastore/Value.java	156	0	2	2	0	0	0	156	154	78

Figure 12



Tests > bookkeeper-server > org.apache.bookkeeper.metastore > Value

Sessions

Value

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
toString()		35%		16%	3	4	8	13	0	1
equals(Object)		42%		37%	4	5	8	13	0	1
hashCode()		0%		0%	2	2	6	6	1	1
merge(Value)		29%		25%	2	3	4	6	0	1
clearFields()		0%		n/a	1	1	2	2	1	1
getFields()		0%		n/a	1	1	1	1	1	1
getFieldsMap()		0%		n/a	1	1	1	1	1	1
project(Set)		100%		100%	0	3	0	8	0	1
Value(Value)		100%		n/a	0	1	0	3	0	1
Value()		100%		n/a	0	1	0	3	0	1
setField(String, byte[])		100%		n/a	0	1	0	2	0	1
getField(String)		100%		n/a	0	1	0	1	0	1
static {...}		100%		n/a	0	1	0	2	0	1
Total	136 of 263	48%	15 of 24	37%	14	25	30	61	4	13

Created with JaCoCo 0.8.4.201905082037

Figure 13 After Partition test

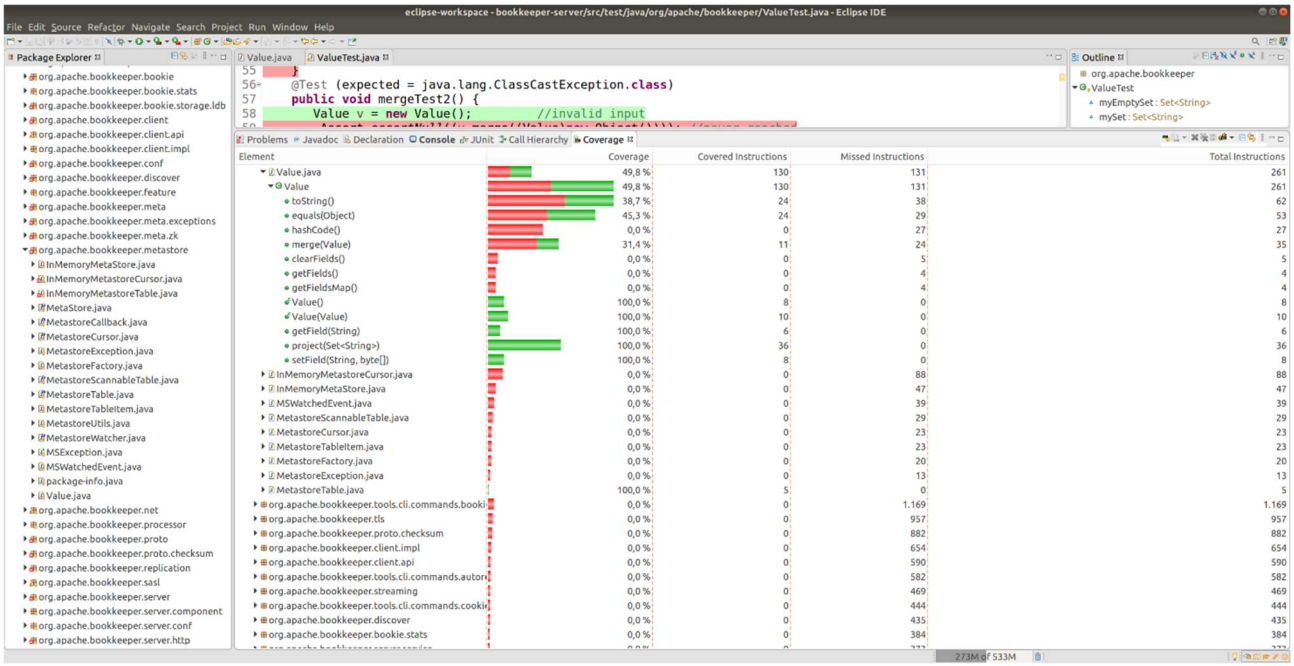


Figure 14 After Partition test (Value)

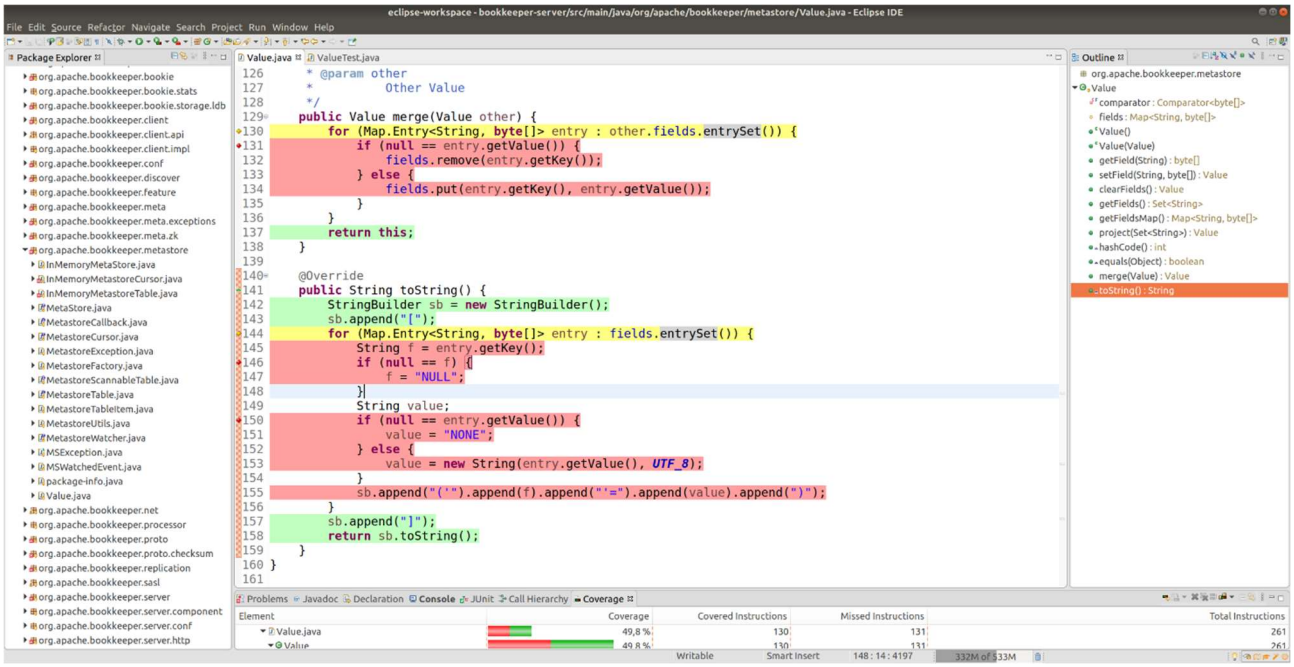


Figure 15 Coverage after Partition test

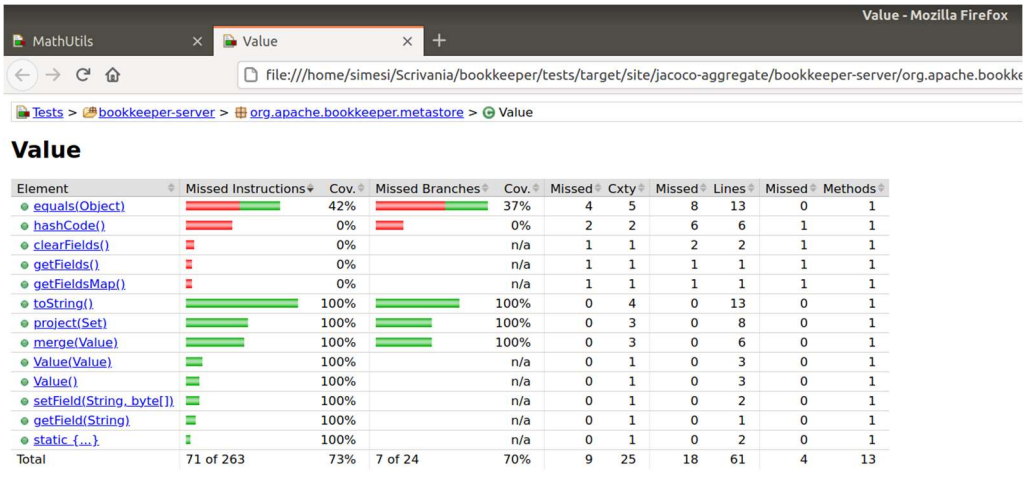


Figure 16 After improvements

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
1	50%	5/10	47%	8/17

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
org.apache.bookkeeper.common.util	1	50%	5/10	47%	8/17

Report generated by PIT 1.5.1

Figure 17 Pit report (Value)

Tests > bookkeeper-common > org.apache.bookkeeper.common.util > MathUtils

MathUtils

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
elapsedMSec(long)		0%		n/a	1	1	1	1	1	1
elapsedMicroSec(long)		0%		n/a	1	1	1	1	1	1
signSafeMod(long, int)		71%		50%	1	2	1	4	0	1
elapsedNanos(long)		0%		n/a	1	1	1	1	1	1
MathUtils()		0%		n/a	1	1	1	1	1	1
nowInNano()		0%		n/a	1	1	1	1	1	1
findNextPositivePowerOfTwo(int)		100%		n/a	0	1	0	1	0	1
Total	25 of 44	43%	1 of 2	50%	6	8	6	10	5	7

Figure 18 After Partition test

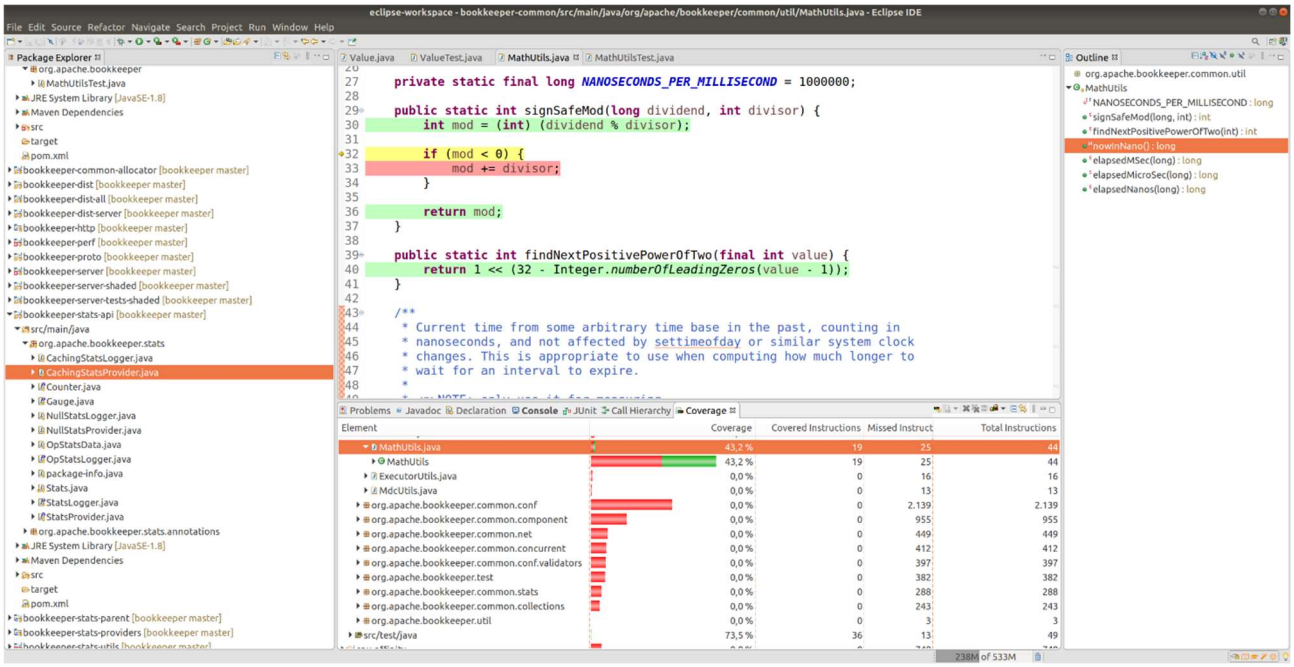


Figure 19 After Partition test

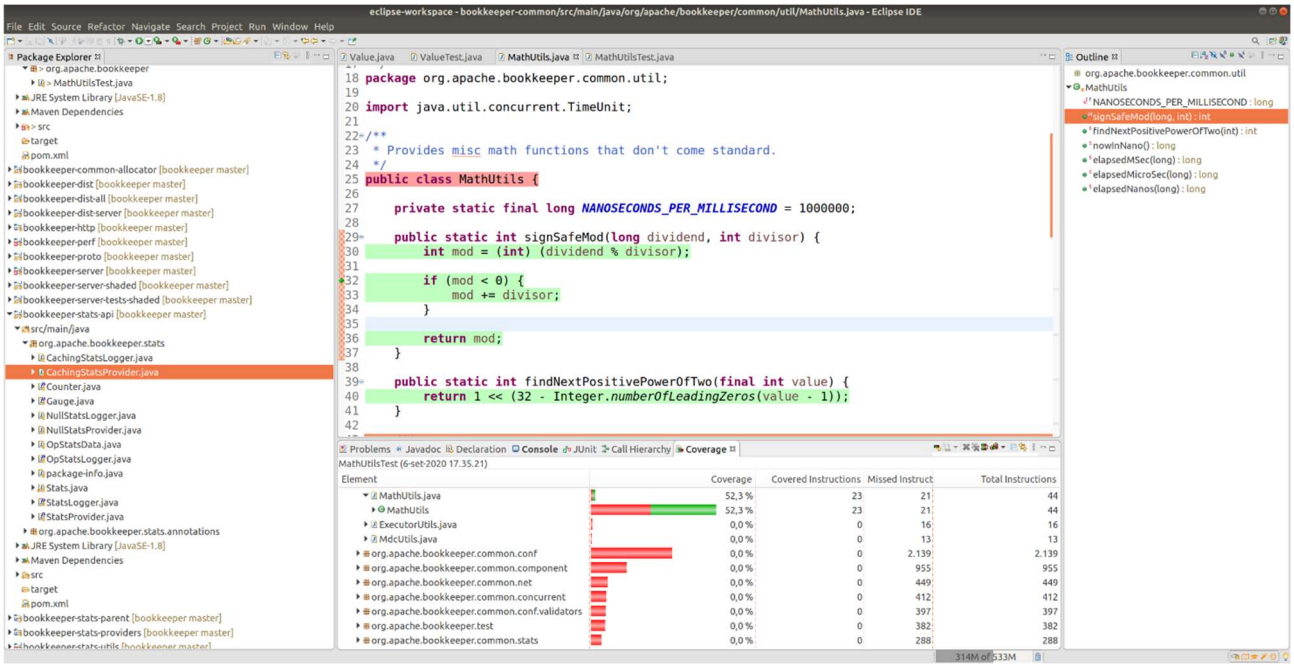


Figure 20 After some improvements

Tests > bookkeeper-common > org.apache.bookkeeper.common.util > MathUtils

MathUtils

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods
elapsedMSec(long)		0%		n/a	1	1	1	1	1	1
elapsedMicroSec(long)		0%		n/a	1	1	1	1	1	1
elapsedNanos(long)		0%		n/a	1	1	1	1	1	1
MathUtils()		0%		n/a	1	1	1	1	1	1
nowInNano()		0%		n/a	1	1	1	1	1	1
signSafeMod(long, int)		100%		100%	0	2	0	4	0	1
findNextPositivePowerOfTwo(int)		100%		n/a	0	1	0	1	0	1
Total	21 of 44	52%	0 of 2	100%	5	8	5	10	5	7

Figure 21 After improvements

Pit Test Coverage Report

Project Summary

Number of Classes	Line Coverage		Mutation Coverage	
1	70%	43/61	50%	11/22

Breakdown by Package

Name	Number of Classes	Line Coverage		Mutation Coverage	
org.apache.bookkeeper.metastore	1	70%	43/61	50%	11/22

Report generated by PIT 1.5.1

Figure 22 Pit report

References

[1] https://bookkeeper.apache.org

[2] http://openjpa.apache.org

[3] https://issues.apache.org/jira/secure/Dashboard.jspa

[4] https://github.com/apache/bookkeeper

[5] https://github.com/apache/openjpa

[6] https://travis-ci.org/

[7] https://sonarcloud.io/

[8] https://junit.org/junit4/

[9] https://www.jacoco.org/jacoco/

[10] https://pitest.org/

[11] https://docs.travis-ci.com/user/sonarcloud/

[12] https://www.eclipse.org/ide/

[13] https://www.virtualbox.org/