

# Indice

<b>1</b>	<b>ComBad</b>	<b>1</b>
1.1	Specifica per il tool . . . . .	1
1.2	Requisiti principali . . . . .	2
1.3	Design dell'architettura software . . . . .	6
1.4	Design di dettaglio . . . . .	6
1.4.1	UML Class Diagram . . . . .	7
1.4.2	UML Sequence Diagram . . . . .	9
1.4.3	UML Component Diagram . . . . .	9
1.5	Progetto della base di dati . . . . .	11
1.6	Graphic User Interface . . . . .	12
1.7	Verifica e validazione dell'approccio . . . . .	12

# Elenco delle figure

1.1	Architettura generale . . . . .	3
1.2	Activity Diagram . . . . .	4
1.3	Use Case Diagram . . . . .	5
1.4	Class Diagram . . . . .	8
1.5	Class Diagram relativo alla GUI . . . . .	9
1.6	Sequence Diagram relativo alla selezione e all'analisi dei dati di un sistema software	10
1.7	Sequence Diagram relativo alla visualizzazione dei committer o dei bad code smell associati . . . . .	11
1.8	Component Diagram . . . . .	11
1.9	Tabella dei bad code smell associati a committer . . . . .	13
1.10	Tabella dei commmitter associati a bad code smell . . . . .	13
1.11	Committer relativi ad un bad code smell . . . . .	14
1.12	Bad code smell relativi ad un committer . . . . .	14

# Elenco delle tabelle

1.1	Principali caratteristiche dei sistemi software analizzati . . . . .	15
1.2	Bad code smell individuati per i sistemi software . . . . .	15
1.3	Numero di coppie <bad code smell, committer> identificate nelle release analizzate di Guava . . . . .	16

# Capitolo 1

## ComBad

### 1.1 Specifica per il tool

Si vuole realizzare una tool-chain di cui il tool ComBad rappresenta l'elemento che permette l'associazione dei bad code smell ai committer. La tool-chain permette di eseguire i seguenti step:

1. selezione di uno dei progetti software open-source ospitati in GitHub, download dei sorgenti di una delle versioni, lettura delle informazioni, registrate in GitHub, relative al progetto selezionato;
2. analisi del codice sorgente per l'individuazione in esso di bad code smell;
3. analisi del file contenente le informazioni relative ai commit per selezionare ed estrarre quelle di interesse;
4. registrazione delle informazioni relative ai commit, committer e bad code smell in una base di dati in modo da consentire successive elaborazioni;
5. analisi ed elaborazione delle informazioni registrate nella base di dati per ottenere le associazioni tra ogni bad code smell e i committer che hanno effettuato un commit sul codice in questione. Un committer associato ad uno o più bad code smell può potenzialmente essere la persona responsabile dell'introduzione dello smell nel codice inficiando la qualità del software.

Per il punto 1 vengono impiegate le funzionalità messe a disposizione da Git e Sourcetree. Per il punto 2 è impiegato il tool di static analysis PMD. Per i punti 3, 4 e 5 è stato sviluppato il tool ComBad che consente di selezionare una versione dei sistemi software scaricati in precedenza da GitHub e per cui sono già stati identificati i bad code smell.

Il tool estrae le informazioni relative ai file sorgenti oggetto di un determinato commit, l'identificativo del committer e il numero delle linee di codice coinvolte nel commit. Queste informazioni sono confrontate con le informazioni relative ad ogni code smell identificato, attraverso il numero delle linee di codice. Per trovare un'associazione vengono confrontati in primis i nomi relativi ai file sorgenti e, se uguali, vengono confrontate anche le linee di codice. Si verifica una corrispondenza se il nome del file individuato da un commit e il nome del file in cui è presente

un bad code smell coincide, e naturalmente se le linee di codice associate ad un commit e ad un code smell si intersecano.

È stato progettato un database relazionale di supporto per ComBad, in cui sono conservate le informazioni prodotte dal tool stesso. In tal modo un utente può effettuare delle interrogazioni alla base di dati, ad esempio per conoscere i committer associati ad un determinato bad code smell oppure, viceversa, conoscere quali code smell sono associati ad un determinato committer. È inoltre possibile anche visualizzare tutti i commit, i committer e i bad code smell relativi al sistema software scelto.

La figura 1.1 riporta l'architettura generale del progetto realizzato: recupero dei file sorgenti da GitHub, analisi del sistema software tramite Sourcetree e PMD per ottenere rispettivamente commit, committer e bad code smell, e utilizzo di ComBad che è supportato da un base di dati.

La figura 1.2 mostra lo UML Activity Diagram modellante l'intero processo eseguito con il supporto della tool-chain.

## 1.2 Requisiti principali

### Casi d'uso

#### [UC1] Seleziona ed analizza un sistema software

**Descrizione caso d'uso:** L'utente seleziona un sistema software che sarà analizzato per associare i code smell precedentemente individuati ai committer operanti su essi

**Attore:** utente

**Input:** nome della versione del sistema software da analizzare

**Precondizioni:** il sistema software selezionato è stato precedentemente analizzato per individuare in esso code smell ed è stato anche scaricato il file registrante i commit effettuati su di esso; quindi devono essere disponibili sia il file dei commit che il file dei bad code smell generato da PMD

**Output:** associazioni tra committer e bad code smell

**Postcondizioni:** lo stato della base di dati è modificato dalla registrazione delle associazioni committer - bad code smell

#### Scenario principale:

1. L'utente seleziona all'interno di una lista di nomi di sistemi software il nome di un sistema o di una particolare versione
2. ComBad effettua il parsing del file con le informazioni relative ai commit e ai committer e dei file con le informazioni relative ai bad code smell, estrae da essi le informazioni necessarie a creare le associazioni tra committer e bad code smell.

#### Scenario alternativo:

2. ComBad preleva le informazioni richieste dall'utente direttamente dalla base di dati, in quanto già elaborate a seguito di una precedente richiesta.

#### [UC2] Visualizza committer e bad code smell associati

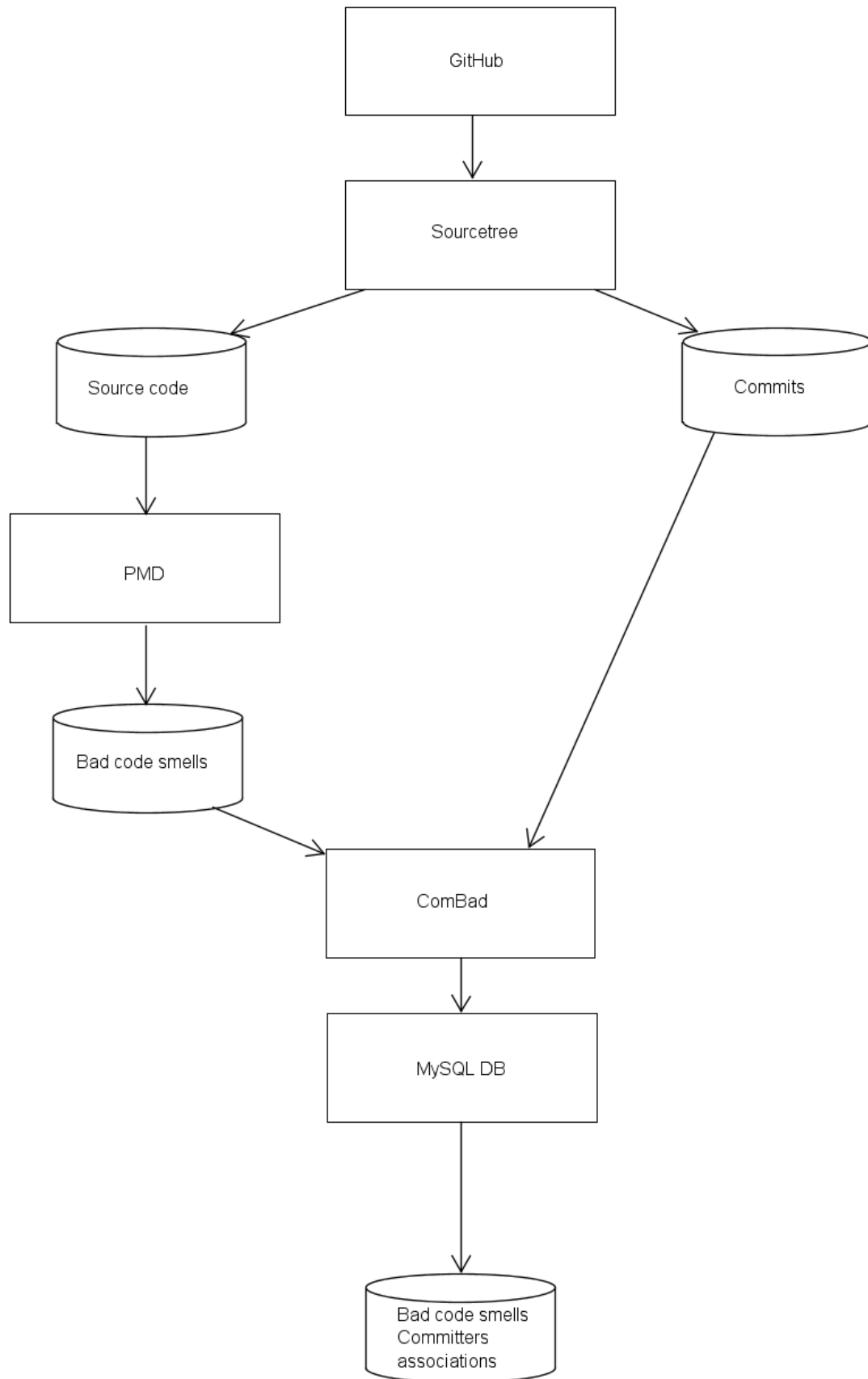


Figura 1.1: Architettura generale

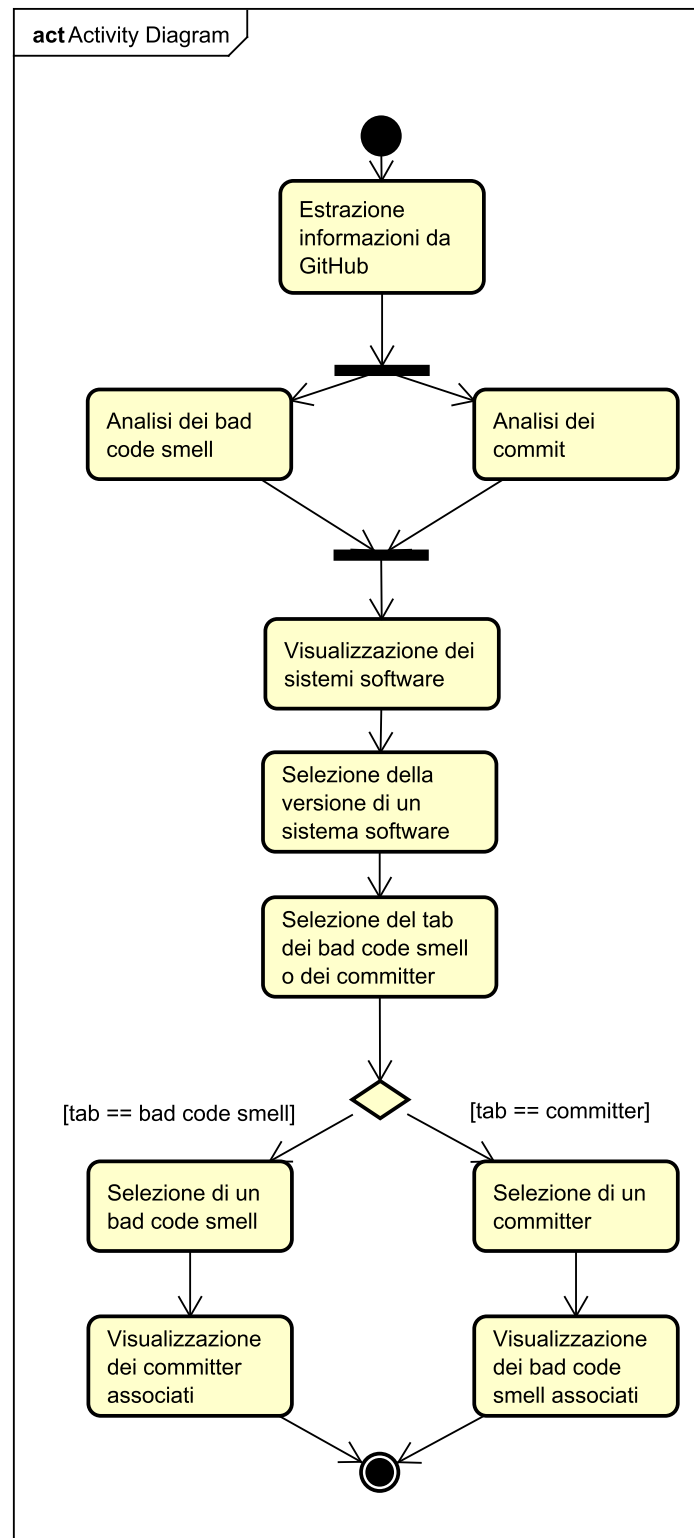


Figura 1.2: Activity Diagram

## 1.2. Requisiti principali

---

**Descrizione caso d'uso:** È visualizzata all'utente la lista dei committer e dei bad code smell ad essi associati per una versione di un sistema software

**Attore:** utente

**Input:** bad code smell o committer di cui visualizzare le associazioni

**Precondizioni:** basi di dati popolata dalle informazioni da prelevare relative alle associazioni

**Output:** lista dei bad code smell associati ad un committer oppure lista dei committer associati ad un bad code smell

**Postcondizioni:** lo stato della base di dati è invariato

**Scenario principale:**

1. *inclusione* di UC1, se non è già stato eseguito
2. Il sistema mostra la lista dei bad code smell individuati nel sistema selezionato
3. L'utente seleziona un bad code smell nella lista
4. ComBad interroga la base di dati producendo la lista dei committer associati al bad code smell selezionato.

**Scenario alternativo:**

2. Il sistema mostra la lista dei committer che hanno operato sul sistema selezionato
3. L'utente seleziona un committer nella lista
4. ComBad interroga la base di dati producendo la lista dei bad code smell associati al committer selezionato.

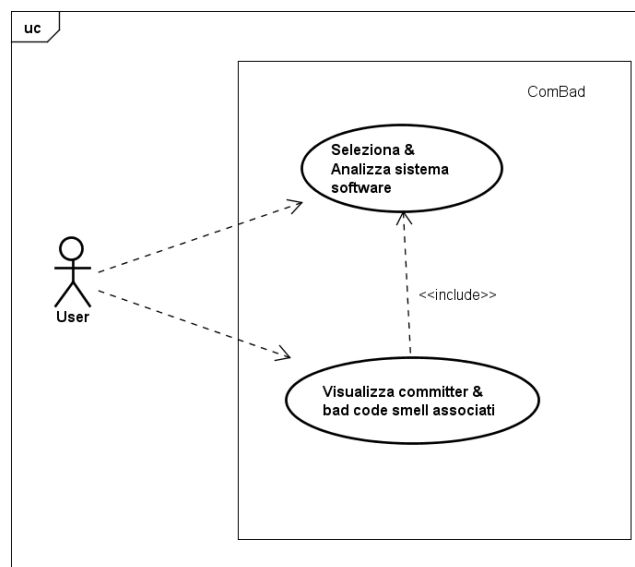


Figura 1.3: Use Case Diagram



### 1.3 Design dell'architettura software

#### Model-view-controller

L'architettura di ComBad prevede l'impiego del pattern architetturale MVC (Model-View-Controller), essendo destinato all'interazione con l'utente. Il pattern MVC è adatto per applicazioni interattive, in quanto fornisce un adeguato disaccoppiamento tra le componenti che interagiscono con l'utente e il resto dell'architettura software.

Nello specifico, l'architettura software si compone di tre parti:

- **Model:** rappresenta il modello dei dati di interesse dell'applicazione ed è rappresentato dalle classi le cui istanze effettuano le elaborazioni che devono essere visualizzate;
- **View:** fornisce una rappresentazione del Model ed è rappresentato dalle classi le cui istanze sono utilizzate per visualizzare i dati del Model;
- **Controller:** definisce la logica di controllo e le funzionalità del sistema, ed è rappresentato dalle classi le cui istanze controllano il Model e le interazioni dell'utente con le View.

### 1.4 Design di dettaglio

La progettazione di ComBad ha richiesto l'utilizzo di alcuni *design pattern GoF*, quali il Singleton e il Facade.

#### Singleton

Il Singleton è un design pattern GoF di tipo creazionale, applicabile ad istanze di classi. È utilizzato per permettere che esista una ed una sola istanza di una classe, ovvero di prevenire che un oggetto sia istanziato più di una volta nell'esecuzione di un programma, fornendo un punto globale di accesso ad esso.

Il Singleton viene utilizzato all'interno del sistema software per garantire che esista una sola istanza della classe impiegata come Controller del sistema e della classe che opera direttamente sulla base di dati.

#### Facade

Il Facade è un design pattern GoF di tipo strutturale, applicabile ad istanze di classi. Fornisce un'interfaccia unificata ad un insieme di interfacce di un sottosistema e definisce un'interfaccia di più alto livello che rende il sottosistema più facile da utilizzare.

Il Facade è impiegato per unificare le interfacce relative agli oggetti che gestiscono i commit, i committer, i bad code smell e le associazioni in rispettive interfacce unificate, quali le classi di gestione dei commit, dei bad code smell e delle associazioni che forniscono interfacce di più alto livello. In tal modo si ottiene una closed architecture e i diversi sottosistemi sono acceduti direttamente dal Controller, il quale non conosce interamente tutte le interfacce che costituiscono quelle unificate.

### 1.4.1 UML Class Diagram

Vengono riportate di seguito delle descrizioni sintetiche relative al sottoinsieme di classi che costituiscono il Model dello UML Class Diagram nella figura 1.4:

- **Commit:** rappresenta i commit effettuati per il sistema software analizzato ed è caratterizzata da attributi quali: identificativo, data, descrizione e committer;
- **Committer:** rappresenta i committer che hanno effettuato un commit sul sistema software analizzato ed è caratterizzata da attributi; quali: nome ed email
- **Change:** rappresenta le operazioni di modifica che sono eseguite in un commit ed è caratterizzata da attributi quali: identificativo, identificativo del commit e file che ha subito la modifica;
- **Range:** rappresenta l'intervallo delle linee di codice di un file sorgente cui è stato applicato un change in un commit ed è caratterizzata da attributi quali: identificativo, identificativo del change, numero di linea di codice iniziale e numero di linea di codice finale;
- **BadCodeSmell:** rappresenta un'interfaccia per i tipi di bad code smell analizzati, quali Dead Code, Duplicated Code, Large Class, Long Method e Long Parameter List. È caratterizzata da attributi comuni ai bad code smell quali: nome della classe di appartenenza, nome del package di appartenenza, nome del file di appartenenza, numero della linea di codice iniziale e finale;
- **DeadCode:** rappresenta le occorrenze di Dead Code individuati in un sistema software ed estende i campi della classe astratta BadCodeSmell definendo un campo per il tipo di Dead Code e per la variabile coinvolta;
- **DuplicatedCode:** rappresenta le occorrenze di Duplicated Code individuati in un sistema software ed estende i campi della classe astratta BadCodeSmell definendo un campo per l'intervallo delle linee di codice clonate;
- **Duplication:** rappresenta un insieme di occorrenze dello stesso Duplicated Code, o meglio una classe di cloni, che vengono individuati nel sistema software e che possono appartenere anche allo stesso file;
- **LargeClass:** rappresenta le occorrenze di Large Class individuate in un sistema software;
- **LongMethod:** rappresenta le occorrenze di Long Method individuati in un sistema software ed estende i campi della classe astratta BadCodeSmell definendo un campo per il nome del metodo;
- **LongParameterList:** rappresenta le occorrenze di Long Parameter List individuati in un sistema software ed estende i campi della classe astratta BadCodeSmell definendo un campo per il nome del metodo.

La figura 1.4 riporta lo UML Class Diagram relativo all'intero sistema software, mentre la figura 1.5 fa riferimento allo UML Class Diagram relativo alla Graphic User Interface di ComBad.

#### 1.4. Design di dettaglio

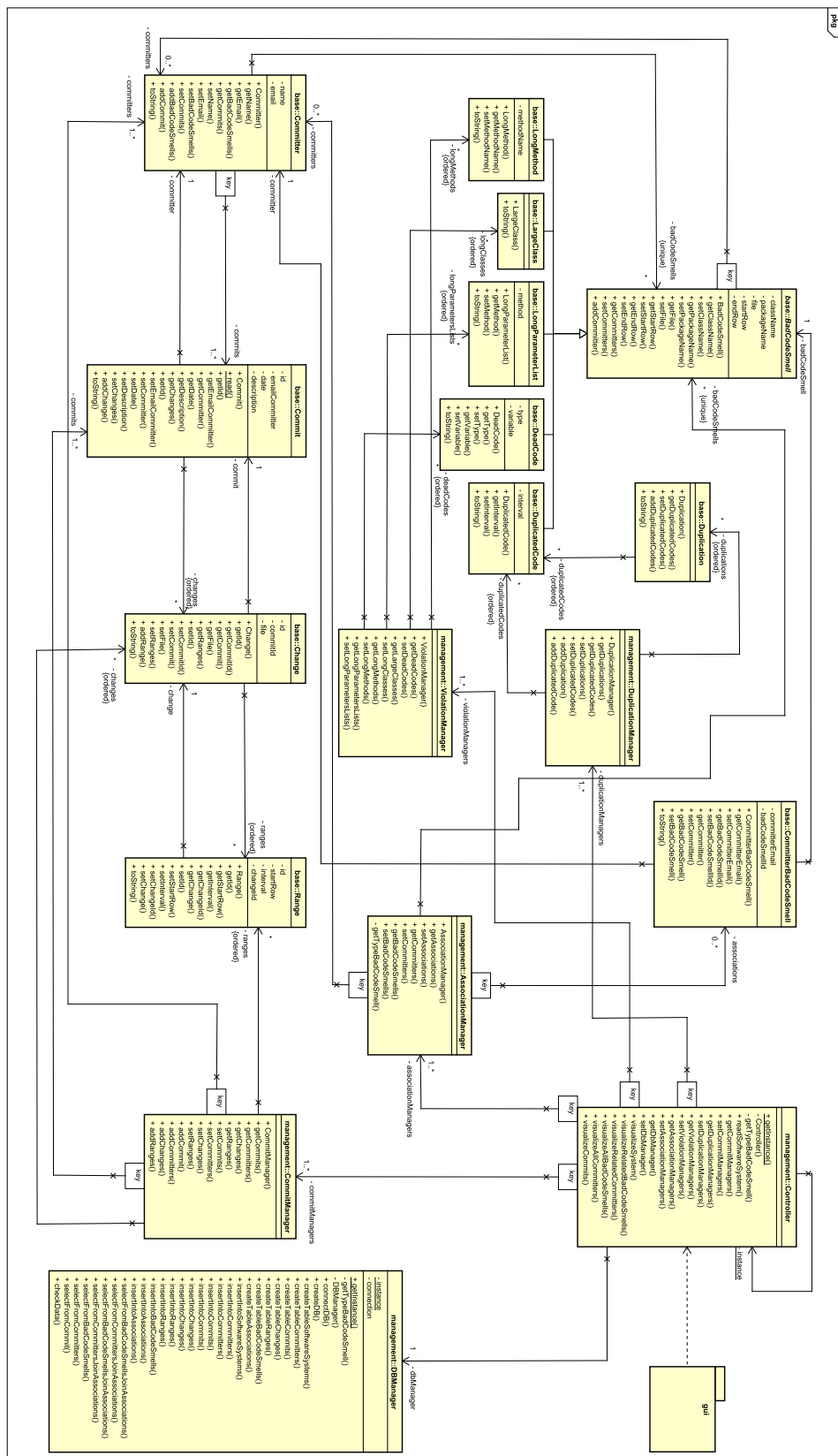


Figura 1.4: Class Diagram

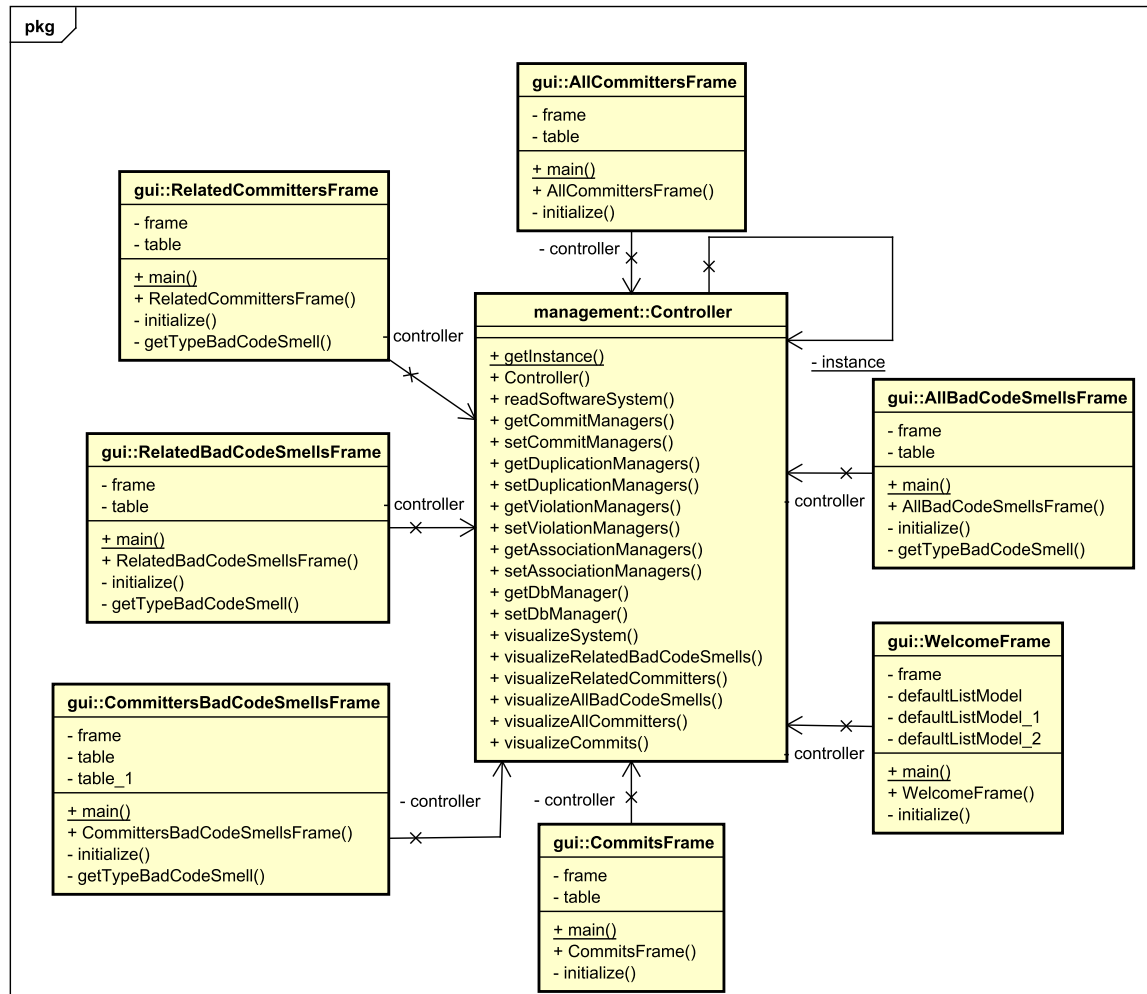


Figura 1.5: Class Diagram relativo alla GUI

### 1.4.2 UML Sequence Diagram

La figura 1.6 riporta lo UML Sequence Diagram relativo al primo caso d'uso che descrive la selezione e l'analisi di un sistema software da parte di ComBad e la presentazione dei risultati all'utente.

La figura 1.7 riporta lo UML Sequence Diagram relativo al secondo caso d'uso che descrive l'interazione da parte dell'utente con ComBad per la visualizzazione dei committer o dei bad code smell associati.

### 1.4.3 UML Component Diagram

La figura 1.8 riporta lo UML Component Diagram modellante i principali componenti di ComBad: il package *management* che include le classi di gestione, il package *base* che raggruppa l'insieme delle classi base e il package *gui* che costituisce le classi relative alla graphic user interface.

ComBad prevede la connessione a MySQL<sup>1</sup> mediante JDBC<sup>2</sup>, un connettore per database

<sup>1</sup><https://www.mysql.com/it/>

<sup>2</sup><https://www.oracle.com/technetwork/java/javase/jdbc/index.html>

## 1.4. Design di dettaglio

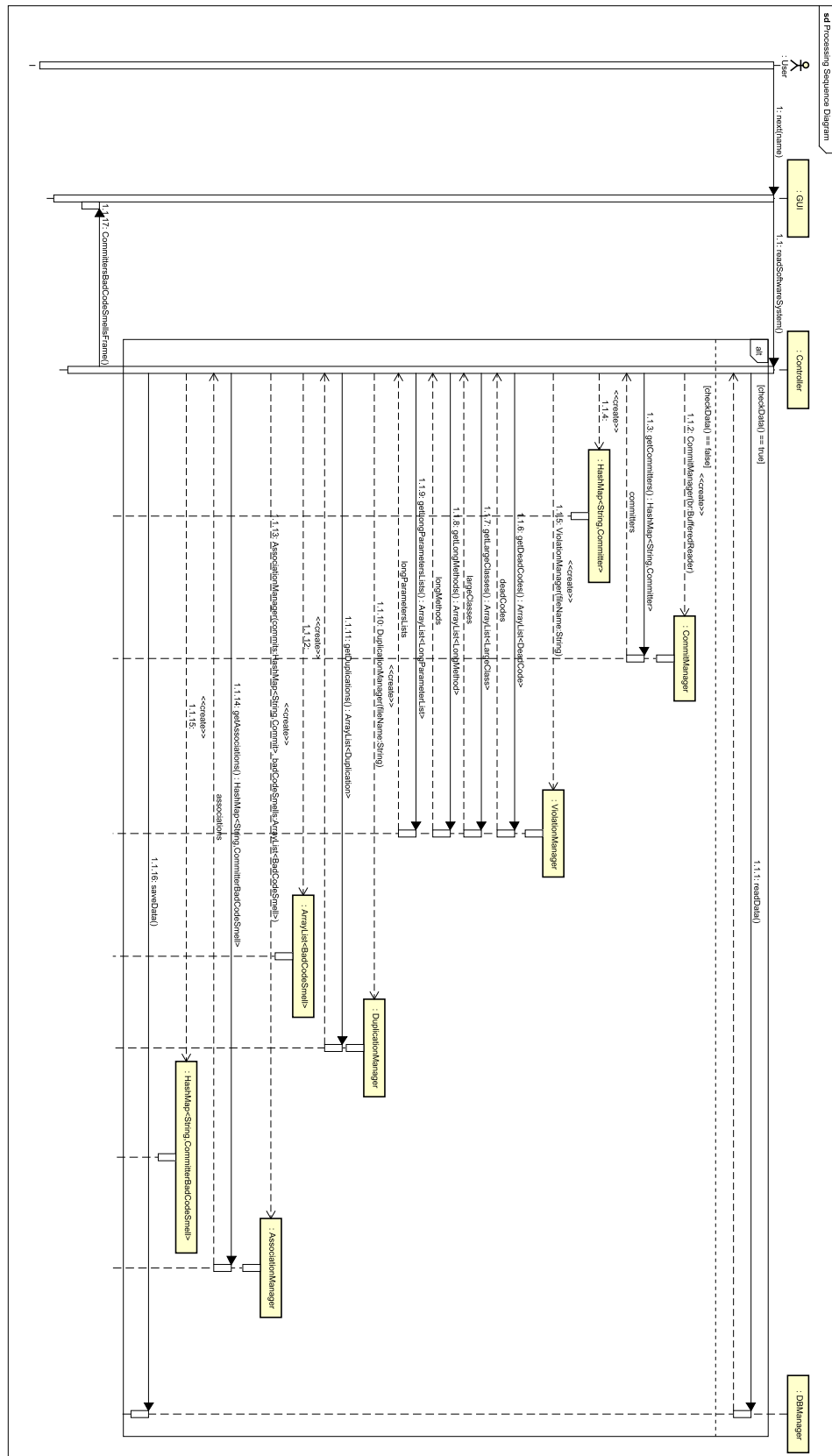


Figura 1.6: Sequence Diagram relativo alla selezione e all'analisi dei dati di un sistema software

## 1.5. Progetto della base di dati

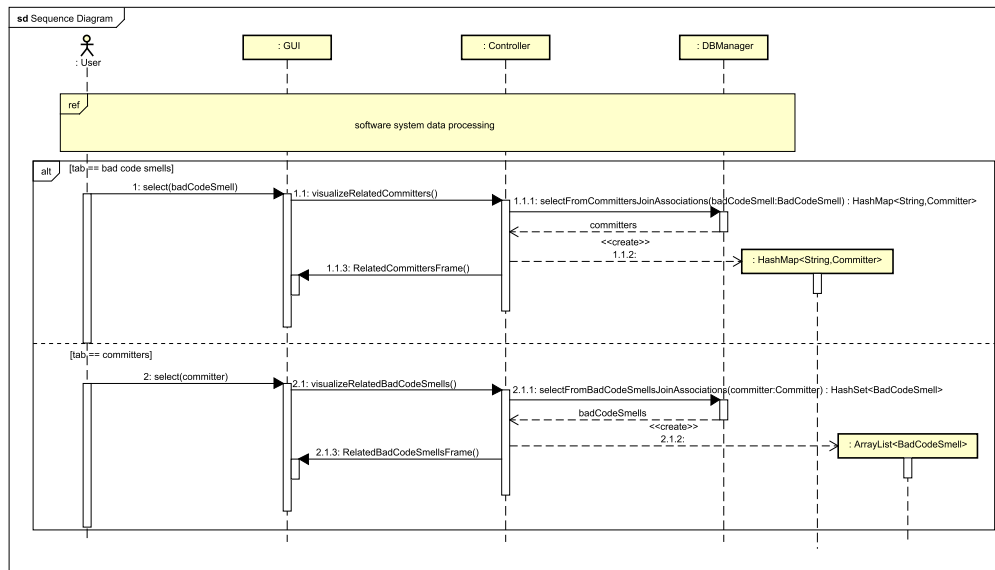


Figura 1.7: Sequence Diagram relativo alla visualizzazione dei committer o dei bad code smell associati

che consente l'accesso e la gestione della persistenza dei dati sulle basi di dati da qualsiasi programma scritto con il linguaggio Java, indipendentemente dal tipo di DBMS utilizzato.

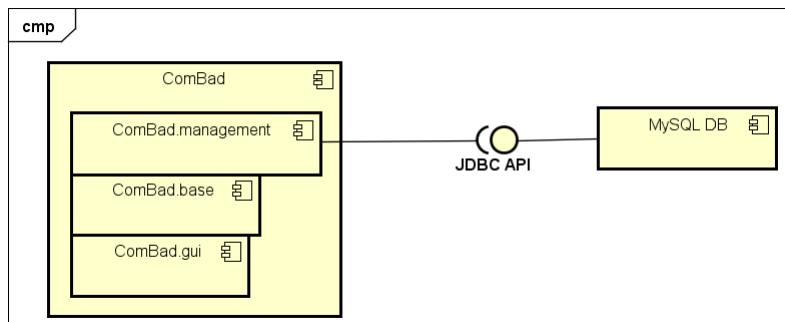


Figura 1.8: Component Diagram

## 1.5 Progetto della base di dati

### Modello relazionale

Si riportano le relazioni che costruiscono le tabelle della base di dati su cui vengono memorizzate le informazioni relative alle associazioni tra committer e bad code smell.

software\_system  $\equiv$  {name, version}

commits  $\equiv$  {commit\_id, committer\_email, date, description, software\_system}

committers  $\equiv$  {email, name}

changes  $\equiv$  {change\_id, commit\_id, file}

ranges  $\equiv$  {range\_id, change\_id, start\_row, interval}

badcodesmells  $\equiv$  {badcodesmell\_id, class\_name, method\_name, package\_name, file, type, va-

```
riable, start_row, end_row, software_system}
associations ≡ {committer_id, badcodesmell_id}
```

La classe DBManager è responsabile della creazione e dell'inserimento dei dati all'interno del database MySQL.

## 1.6 Graphic User Interface

Vengono riportate di seguito delle figure relative alla graphic user interface. All'avvio di Combad, è presentata la prima interfaccia grafica dalla quale è possibile selezionare una versione di un sistema software (figura ??).

Dopo aver selezionato un sistema software, ComBad procede con l'elaborazione dei dati relativi al sistema software scelto (figura ??).

Al termine dell'elaborazione dei dati, ComBad mostra un'interfaccia in cui sono presenti due tab. Il primo riporta una tabella dei bad code smell associati a committer del sistema software scelto (figura 1.9). L'altro mostra una tabella dei committer associati ai bad code smell individuati. Per ogni committer sono mostrati il numero totale di bad code smell associati e un numero per ogni tipo dei code smell individuato (figura 1.10).

L'utente, se seleziona un bad code smell, richiede la lista dei committer associati ad esso, la quale viene mostrata da ComBad in una nuova tabella (figura 1.11). Se invece l'utente seleziona un committer, richiede l'elenco dei bad code smell associati ad esso, il quale viene mostrato da ComBad in una nuova tabella (figura 1.12).

Cliccando tra le varie voci presenti nel menù a tendina "Visualize", ComBad offre la possibilità di visualizzare tutti i bad code smell individuati e tutti i committer del sistema software scelto anche se questi non sono associati tra loro. Inoltre è possibile visualizzare tutti i commit che sono stati effettuati nel repository del sistema.

## 1.7 Verifica e validazione dell'approccio

Per verificare e validare l'approccio definito, il tool ComBad è stato utilizzato in alcuni esperimenti. Sono stati selezionati tre progetti software presenti in GitHub: dnsjava<sup>3</sup> (v2.1.9), Apache Tika<sup>4</sup> (1.21) e Guava<sup>5</sup>, per il quale sono state analizzate tre differenti release (v27.0, v27.1, v28.0).

La tabella 1.1 riporta alcune caratteristiche dimensionali di tali sistemi software.

### Dnsjava

Dnsjava è un'implementazione di un protocollo DNS, completamente realizzato in Java.

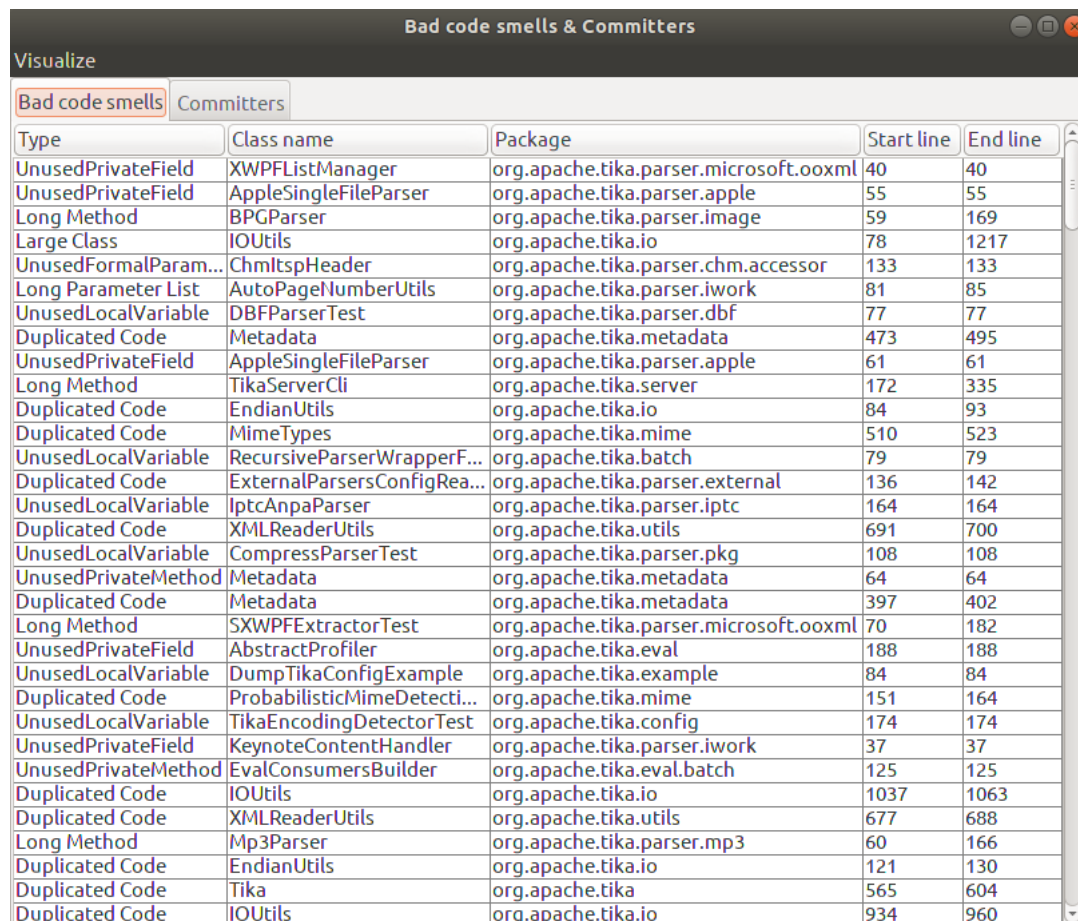
---

<sup>3</sup><https://github.com/dnsjava>

<sup>4</sup><https://github.com/apache/tika>

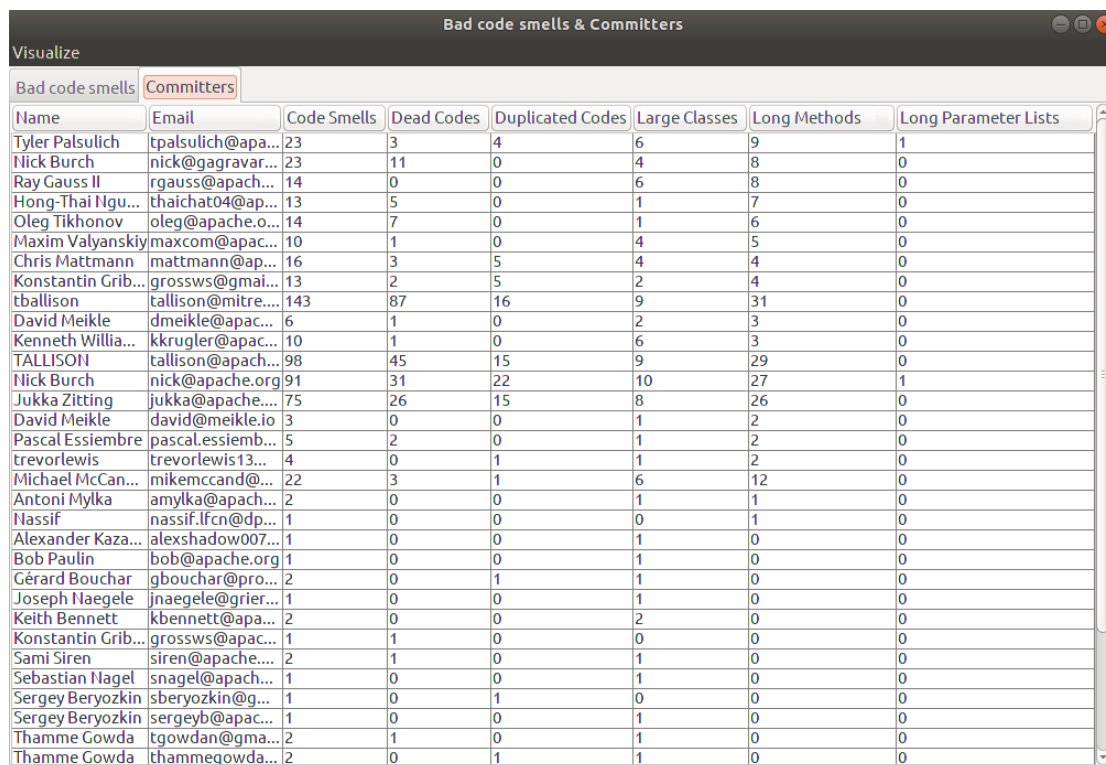
<sup>5</sup><https://github.com/google/guava>

## 1.7. Verifica e validazione dell'approccio



Type	Class name	Package	Start line	End line
UnusedPrivateField	XWPFListManager	org.apache.tika.parser.microsoft.ooxml	40	40
UnusedPrivateField	AppleSingleFileParser	org.apache.tika.parser.apple	55	55
Long Method	BPGParser	org.apache.tika.parser.image	59	169
Large Class	IOUtils	org.apache.tika.io	78	1217
UnusedFormalParam...	ChmItspHeader	org.apache.tika.parser.chm.accessor	133	133
Long Parameter List	AutoPageNumberUtils	org.apache.tika.parser.iwork	81	85
UnusedLocalVariable	DBFParserTest	org.apache.tika.parser.dbf	77	77
Duplicated Code	Metadata	org.apache.tika.metadata	473	495
UnusedPrivateField	AppleSingleFileParser	org.apache.tika.parser.apple	61	61
Long Method	TikaServerCli	org.apache.tika.server	172	335
Duplicated Code	EndianUtils	org.apache.tika.io	84	93
Duplicated Code	MimeTypes	org.apache.tika.mime	510	523
UnusedLocalVariable	RecursiveParserWrapperF...	org.apache.tika.batch	79	79
Duplicated Code	ExternalParsersConfigRea...	org.apache.tika.parser.external	136	142
UnusedLocalVariable	IptcAnpaParser	org.apache.tika.parser.iptc	164	164
Duplicated Code	XMLReaderUtils	org.apache.tika.utils	691	700
UnusedLocalVariable	CompressParserTest	org.apache.tika.parser.pkg	108	108
UnusedPrivateMethod	Metadata	org.apache.tika.metadata	64	64
Duplicated Code	Metadata	org.apache.tika.metadata	397	402
Long Method	SXWPFFExtractorTest	org.apache.tika.parser.microsoft.ooxml	70	182
UnusedPrivateField	AbstractProfiler	org.apache.tika.eval	188	188
UnusedLocalVariable	DumpTikaConfigExample	org.apache.tika.example	84	84
Duplicated Code	ProbabilisticMimeDetecti...	org.apache.tika.mime	151	164
UnusedLocalVariable	TikaEncodingDetectorTest	org.apache.tika.config	174	174
UnusedPrivateField	KeynoteContentHandler	org.apache.tika.parser.iwork	37	37
UnusedPrivateMethod	EvalConsumersBuilder	org.apache.tika.eval.batch	125	125
Duplicated Code	IOUtils	org.apache.tika.io	1037	1063
Duplicated Code	XMLReaderUtils	org.apache.tika.utils	677	688
Long Method	Mp3Parser	org.apache.tika.parser.mp3	60	166
Duplicated Code	EndianUtils	org.apache.tika.io	121	130
Duplicated Code	Tika	org.apache.tika	565	604
Duplicated Code	IOUtils	org.apache.tika.io	934	960

Figura 1.9: Tabella dei bad code smell associati a committer



Name	Email	Code Smells	Dead Codes	Duplicated Codes	Large Classes	Long Methods	Long Parameter Lists
Tyler Palsulich	tpalsulich@apa...	23	3	4	6	9	1
Nick Burch	nick@gagravar...	23	11	0	4	8	0
Ray Gauss II	rgauss@apach...	14	0	0	6	8	0
Hong-Thai Ngu...	thaichat04@ap...	13	5	0	1	7	0
Oleg Tikhonov	oleg@apache.o...	14	7	0	1	6	0
Maxim Valyanskiy	maxcom@apac...	10	1	0	4	5	0
Chris Mattmann	mattmann@ap...	16	3	5	4	4	0
Konstantin Grib...	grossws@gmal...	13	2	5	2	4	0
tballison	tallison@mitre...	143	87	16	9	31	0
David Meikle	dmeikle@apac...	6	1	0	2	3	0
Kenneth Willia...	kkrugler@apac...	10	1	0	6	3	0
TALLISON	tallison@apach...	98	45	15	9	29	0
Nick Burch	nick@apache.org	91	31	22	10	27	1
Jukka Zitting	jukka@apache....	75	26	15	8	26	0
David Meikle	david@meikle.io	3	0	0	1	2	0
Pascal Essiembre	pascal.essiem...	5	2	0	1	2	0
trevorlewis	trevorlewis13...	4	0	1	1	2	0
Michael McCan...	mikemccand@...	22	3	1	6	12	0
Antoni Mylka	amylka@apach...	2	0	0	1	1	0
Nassif	nassif.lfcdp...	1	0	0	0	1	0
Alexander Kaza...	alexshadow007...	1	0	0	1	0	0
Bob Paulin	bob@apache.org	1	0	0	1	0	0
G�rard Bouchar	gbouchar@pro...	2	0	1	1	0	0
Joseph Naegele	jnaegele@grier...	1	0	0	1	0	0
Keith Bennett	kbennett@apa...	2	0	0	2	0	0
Konstantin Grib...	grossws@apac...	1	1	0	0	0	0
Sami Siren	siren@apache....	2	1	0	1	0	0
Sebastian Nagel	snagel@apach...	1	0	0	1	0	0
Sergey Beryozkin	sberozkin@g...	1	0	1	0	0	0
Sergey Beryozkin	sergeyb@apac...	1	0	0	1	0	0
Thamme Gowda	tgowdan@gma...	2	1	0	1	0	0
Thamme Gowda	thammegowda...	2	0	1	1	0	0

Figura 1.10: Tabella dei committer associati a bad code smell



## 1.7. Verifica e validazione dell'approccio

The screenshot shows the 'Bad code smells & Committers' application. The 'Committers' tab is selected. A 'Related committers' dialog box is open, displaying a list of committers and their email addresses. The main table lists various bad code smells, their types, class names, packages, and line ranges.

Type	Class name	Package	Start line	End line
Duplicated Code	Metadata	org.apache.tika.metadata	397	402
Long Method	SXWPExtractorTest	org.apache.tika.parser.microsoft.ooxml	70	182
UnusedPrivateField	AbstractProfiler	org.apache.tika.eval	188	188
UnusedLocalVariable	DumpTikaConfigExample	org.apache.tika.example	84	84
Duplicated Code	ProbabilisticMimeDetecti...	org.apache.tika.mime	151	164
UnusedLocalVariable	TikaEncodingDetectorTest	org.apache.tika.config	174	174
UnusedPrivateField	KeynoteContentHandler	org.apache.tika.parser.iwork	37	37
UnusedPrivateMethod	EvalConsumersBuilder	org.apache.tika.eval.batch	125	125
Duplicated Code	IOUtils	org.apache.tika.io	1037	1063
Duplicated Code	XMLReaderUtils	org.apache.tika.utils	677	688
Long Method	Mp3Parser	org.apache.tika.parser.mp3	60	166
Duplicated Code	EndianUtils	org.apache.tika.io	121	130
Duplicated Code	Tika	org.apache.tika	565	604
Duplicated Code	IOUtils	org.apache.tika.io	934	960

Name	Email
Tyler Palsulich	tpalsulich@apache.org
Tim Allison	tallison@apache.org
Nick Burch	nick@apache.org

Figura 1.11: Committer relativi ad un bad code smell

The screenshot shows the 'Bad code smells & Committers' application. The 'Committers' tab is selected. A 'Related bad code smells' dialog box is open, displaying a list of bad code smells and their related committers. The main table lists various bad code smells, their types, class names, packages, and line ranges.

Type	Class name	Package	Start line	End line
Duplicated Code	XMLReaderUtils	org.apache.tika.u...	255	263
Long Method	WordExtractor	org.apache.tika.p...	266	381
UnusedLocalVari...	RecursiveParser...	org.apache.tika.p...	341	341
UnusedLocalVari...	DBFParserTest	org.apache.tika.p...	77	77
UnusedPrivateField	HtmlParser	org.apache.tika.p...	59	59
UnusedLocalVari...	AnalyzerManager...	org.apache.tika.e...	90	90
Long Method	OOXMLContainer...	org.apache.tika.p...	140	277
UnusedPrivateField	AbstractDBParser	org.apache.tika.p...	45	45
UnusedPrivateMe...	EvalConsumersB...	org.apache.tika.e...	125	125
Long Method	XWPFWordExtrac...	org.apache.tika.p...	183	346
UnusedPrivateField	EvalConsumerBui...	org.apache.tika.e...	46	46
UnusedPrivateField	PackageParser	org.apache.tika.p...	89	89
Duplicated Code	ForkClient	org.apache.tika.f...	140	146
UnusedPrivateField	AppleSingleFileP...	org.apache.tika.p...	64	64
UnusedLocalVari...	DBFCell	org.apache.tika.p...	138	138
UnusedFormalPa...	CharsetMatch	org.apache.tika.p...	69	69
UnusedPrivateField	AppleSingleFileP...	org.apache.tika.p...	62	62
UnusedLocalVari...	RecursiveParser...	org.apache.tika.p...	340	340
UnusedPrivateMe...	NonDetectingEnc...	org.apache.tika.d...	53	53
Long Method	OOXMLWordAnd...	org.apache.tika.p...	202	356
UnusedPrivateMe...	ParseContext	org.apache.tika.p...	180	180
Long Method	TikaCLI	org.apache.tika.cli	358	498
UnusedPrivateField	XWPFWordExtrac...	org.apache.tika.p...	75	75
Long Method	TextExtractor	org.apache.tika.p...	1034	1318
UnusedLocalVari...	DigestingParserT...	org.apache.tika.p...	135	135
UnusedLocalVari...	DBFParserTest	org.apache.tika.p...	88	88

Name	Email	Code Smells	Dead Codes	Duplicated Codes	Large Classes	Long Methods	Long Parameter Lists
Sergey Pilyuzkin	sergey@apache...	1	0	0	0	0	0
Thamme Gowda	tgowdan@gma...	1	0	1	0	0	0
Thamme Gowda	thammegowda...	2	0	1	0	0	0

Figura 1.12: Bad code smell relativi ad un committer

## Apache Tika

Apache Tika è un toolkit per il rilevamento e l'estrazione di metadati e contenuti di testo strutturato da vari documenti, utilizzando le librerie di parser esistenti. Tika è un progetto di Apache Software Foundation.

## Guava

Guava è un set di librerie base di Google che include una libreria di grafi, API/utilità per la concorrenza, I/O, hashing, elaborazione di stringhe ed altro.

Sistema software	Release	Numero di classi	Numero di metodi	NLOC
dnsjava	v2.1.9	278	2225	25309
Apache Tika	1.21	1390	8234	134457
Guava	v27.0	7442	57806	511593
Guava	v27.1	7422	57876	512767
Guava	v28.0	7413	57929	513202

Tabella 1.1: Principali caratteristiche dei sistemi software analizzati

## Identificazione dei bad code smell

L'analisi di dnsjava con PMD ha portato all'identificazione di 65 bad code smell, di cui la maggior parte è costituita da Duplicated Code (36). A seguire Dead Code (13), Long Method (8), Long Parameter List (6) e Large Class (2).

Per il sistema Apache Tika, invece, sono stati identificati 360 bad code smell, di cui la maggior parte è costituita da Dead Code (240). A seguire Duplicated Code (62), Long Method (47), Large Class (10) e Long Parameter List (1).

Per quanto riguarda il sistema software Guava, sono state analizzate le ultime attuali tre release, in modo da poter valutare anche l'evoluzione dei code smell individuati. In particolare, si è notata una crescita di bad code smell seppur limitata. Nella release v27.0 sono presenti 843 code smell di cui 810 Duplicated Code, 18 Large Class, 13 Dead Code e 2 Long Method. Nella release v27.1 vi è un incremento di 34 Duplicated Code, 1 Large Class e 1 Long Parameter List rispetto alla release precedente, per un totale di 879 bad code smell. Nella release v28.0 è presente

Sistema Software	Release	Bad code smell	Dead Code	Duplicated Code	Large Class	Long Method	Long Parameter List
dnsjava	v2.1.9	65	13	36	2	8	6
Apache Tika	1.21	360	240	62	10	47	1
Guava	v27.0	843	13	810	18	2	0
Guava	v27.1	879	13	844	19	2	1
Guava	v28.0	888	13	851	21	2	1

Tabella 1.2: Bad code smell individuati per i sistemi software

ancora un aumento di 7 Duplicated Code e 2 Large Class rispetto alla release precedente, per un totale di 888 bad code smell.

Naturalmente il numero di bad code smell è fortemente influenzato dalle dimensioni del sistema software, in particolare, non solo dalle linee di codice, ma anche dal numero di classi e di metodi. Come è possibile notare nella tabella 1.2, Guava presenta un numero di code smell molto superiore agli altri sistemi presi in considerazione, viste le sue dimensioni rispetto a quelle degli altri due.

### Analisi dei commit e dei committer

Per il progetto software dnsjava sono stati considerati tutti i commit effettuati. Al momento sono 1748 e sono stati effettuati da 8 committer. Anche per Apache Tika sono stati considerati tutti i commit effettuati. Questi risultano essere 4495 ed effettuati da 108 committer.

Per quanto riguarda Guava, l'analisi dei commit e dei committer è stata effettuata solo per un numero limitato di essi, in particolare per i commit introdotti tra il rilascio di una release e l'altra. Ciò è stato effettuato al fine di capire se i committer tra una release e l'altra sono maggiormente propensi ad effettuare operazioni di refactoring e/o re-engineering, oppure semplicemente ad apportare solo nuove feature.

Nel dettaglio, per la release v27.0 sono stati considerati 24 commit effettuati da 5 committer, mentre per la release v27.1 sono stati considerati 66 commit effettuati da 5 committer, ed infine per la release v28.0 sono stati considerati 66 commit effettuati da 2 committer.

### Analisi dei risultati

Dall'analisi condotta da ComBad per dnsjava sono state prodotte 65 coppie <bad code smell, committer> per un totale di 2 committer coinvolti. Il numero massimo di code smell associati risulta essere 44, mentre il numero minimo 39.

L'analisi per Apache Tika ha prodotto 360 coppie <bad code smell, committer> per un totale di 39 committer coinvolti. Il numero massimo di code smell associati ad un committer risulta essere 143, mentre il numero minimo 1.

Sistema Software	Release	Bad code smell	Dead Code	Duplicated Code	Large Class	Long Method	Long Parameter List
Guava	v27.0	7	1	3	3	0	0
Guava	v27.1	80	0	68	12	0	0
Guava	v28.0	40	0	36	4	0	0

Tabella 1.3: Numero di coppie <bad code smell, committer> identificate nelle release analizzate di Guava

Per quanto riguarda la release v27.0 di Guava, ComBad ha prodotto solo 7 coppie <bad code smell, committer> per un totale di 5 committer coinvolti. Il numero massimo di code smell associati ad un committer risulta essere 4, mentre il numero minimo 2.

Situazione diversa per la release v27.1 di Guava, per la quale ComBad ha prodotto 80 coppie <bad code smell, committer> per un totale di 5 committer coinvolti. Il numero massimo di code smell associati ad un committer risulta essere 77, mentre il numero minimo 3.

Infine per la release v28.0 di Guava sono state prodotte 40 coppie <bad code smell, committer> per un totale di 2 committer coinvolti. Il numero massimo di code smell associati ad un committer risulta essere 39, mentre il numero minimo 21.

Le varie analisi, condotte per Guava, fanno notare che il numero di bad code smell è aumentato tra un release e l'altra. In particolare, una piccola percentuale di essi coincide tra le varie versioni e questo potrebbe significare che i commit apportati al sistema software non sono relativi a interventi di refactoring e/o re-engineering. Le restanti parti dei code smell di ogni release sono diverse tra loro e ciò potrebbe implicare l'inserimento di nuove feature da parte dei committer. Infine, l'aumento del numero totale dei bad code smell tra le varie release può essere legato all'aumento del numero totale di metodi e di LOC nelle varie release analizzate (anche se è diminuito il numero totale delle classi).

Alcuni bad code smell non è stato possibile associarli a dei committer; questo può dipendere dal fatto che gli smell erano stati introdotti in versioni precedenti (non analizzate negli esperimenti effettuati) e che nessun commit era stato effettuato sulle relative porzioni di codice, da cui l'impossibilità di realizzare le associazioni.

La tabella 1.3 riporta un riepilogo delle informazioni appena descritte.

Tutti i risultati ottenuti negli esperimenti condotti sono stati verificati e validati con ispezione manuale ed essi sono risultati essere corretti per quanto riguarda l'attuale versione del tool ComBad.