Indice

Introduzione ……. 2

Capitolo 1: Cloni software ……... 5

1.1 Introduzione ……. 5

1.2 Tipi di cloni ……. 8

1.3 Tecniche di individuazione cloni ……... 11

1.4 Processo di individuazione cloni ……... 13

Capitolo 2: Il tool di Clone Detection NiCad …… 16

2.1 Principali caratteristiche di NiCad ……. 16

2.2 Applicazione di NiCad …….. 17

Capitolo 3: Il sistema di controllo versione Git e di hosting GitHub ……...…. 20

3.1 Principali caratteristiche di Git ……..…… 20

3.2 Implementazione ……...…… 22

3.3 GitHub ……...…… 24

Capitolo 4: Il tool Sourcetree ……...…….. 25

4.1 Principali caratteristiche di Sourcetree ……..… 25

Capitolo 5: Il tool Clo_tter …… 28

5.1 Definizione caso studio ……. 29

5.2 Descrizione dei principali casi studio ……..…….. 32

5.3 Il Design del tool Clo_tter ……. 35

5.4 Implementazione ……..……. 43

5.5 Applicazione dell'approccio …….. 49

Conclusioni ……...…. 52

Ringraziamenti ……...…… 54

Bibliografia …… 55

Introduzione

Il presente lavoro di tesi si propone di analizzare dati registrati in repository di progetti software open-source per individuare in essi la presenza di cloni software ed associarli ai relativi committers.

Sono detti cloni software coppie di segmenti di codice (sequenze di linee di codice) sorgente simili tra loro secondo una qualche definizione di similarità [Baxter, 2002], da un punto di vista lessicale, sintattico o semantico. Due segmenti di codice, quindi, sono cloni se uno è un duplicato dell'altro (più o meno esatto dal punto di vista lessicale e sintattico), o se presentano una similarità semantica in quanto implementano una stessa funzione (hanno pre- e post-condizioni simili).

La presenza di cloni in un sistema software può inficiarne la qualità, soprattutto per quanto riguarda la sua manutenzione ed evoluzione. Si pensi, ad esempio, al caso in cui è rilevato un errore in un segmento di codice clonato più volte nel sistema: in tal caso la stessa modifica per rimuovere l'errore deve essere applicata a tutti i segmenti clonati. Ciò richiede la precisa conoscenza dell'esistenza dei cloni e della loro localizzazione nel sistema, e comunque comporta un maggior onere della necessaria modifica (che va, appunto, ripetuta per ciascun clone).

In letteratura sono presenti varie metodologie, tecniche e tool per individuare cloni in un sistema software al fine di documentarne e segnalarne la presenza, in modo da poter poi eliminarne o ridurne la presenza attuando adeguate attività di manutenzione, quali ad esempio la loro rifattorizzazione in moduli.

Un aspetto interessante, è quello relativo a capire chi, tra sviluppatori e/o manutentori (i committer) di un sistema, ha introdotto i cloni in esso presenti, col fine di capire se c'è qualche committer maggiormente propenso ad introdurne, se clona proprio codice o quello prodotto da altri, quali possono essere motivazioni inducenti i committer all'introduzione dei cloni. Tale informazione è utile a migliorare la gestione dello sviluppo ed evoluzione sistema.

In riferimento a tale ultimo aspetto, questo lavoro di tesi propone un approccio che permette di effettuare l'associazione tra cloni individuati in un sistema ed i possibili committer che li hanno introdotti.

A tal fine si sono utilizzate le informazioni registrate in un repository per la gestione di sistemi sw open-source. In particolare, dal repository si sono scaricati i file sorgenti di una versione di un sistema software, e le informazioni relative ai vari commit per essi riportanti i rispettivi committers.

I file sorgenti scaricati sono state analizzati da un tool per l'individuazione di cloni in essi e le informazioni ottenute da tale analisi sono state 'incrociate' con quelle relative ai commit e committers, in modo da poter associare ciascun clone al committer che per ultimo aveva manipolato il file e quindi poter essere (l'ultimo) responsabile dell'introduzione del clone.

Si è fatto riferimento a GitHub quale sistema di hosting di sistemi open-source da cui scaricare un sistema software e le informazioni relative a commit e committers, mentre si è utilizzato il tool NiCad per l'individuazione dei cloni.

È stato quindi sviluppato un tool, chiamato Clo_tter, per effettuare l'analisi e la sintesi delle informazioni estratte da GitHub ed associarle a quelle ottenute da NiCad; tale tool memorizza le informazioni così elaborate in un data base che può essere interrogato per ottenere le associazioni tra cloni e committers.

Sono stati effettuati alcuni esperimenti utilizzando sistemi software open source prelevati di GitHub per verificare e validare il tool Clo_tter, che è risultato rispondere in pieno ai requisiti per esso specificati.

Relativamente alle informazioni prodotte dal tool relative all'associazione tra i cloni identificati ed i committer, esse sono un primo utile risultato per le successive analisi da effettuare per individuare i possibili committer che hanno effettivamente introdotto quei cloni.

La tesi è strutturata secondo i seguenti capitoli:

Capitolo 1: Cloni software; il capitolo riporta i concetti relativi ai cloni software ed alla loro identificazione.

Capitolo 2: Il tool di Clone Detection NiCad; è descritto il tool NiCad usato per la identificazione di cloni software nei sistemi

Capitolo 3: Il sistema di controllo di versione Git; è descritta la struttura del sistema di hosting di sistemi open source GitHub

Capitolo 4: Il tool Sourcetree; è descritto descrive il tool Sourcetree utilizzato per prelevare le informazioni di interesse da GitHub.

Capitolo 5: Il tool Clo_tter; è presentato il tool sviluppato per effettuare l'associazione tra committer e cloni

Capitolo 6: Conclusioni; riporta le principali conclusioni del lavoro svolto e possibili sviluppi futuri.

Capitolo 1: Cloni software

1.1 Introduzione

Sono detti cloni software coppie di segmenti di codice (sequenze di linee di codice) sorgente simili tra loro secondo una qualche definizione di similarità [Baxter], [Koske] da un punto di vista lessicale, sintattico o semantico. In un sistema software è possibile avere cloni software all'interno di uno stesso programma o tra programmi/moduli differenti.

Due segmenti di codice, quindi, sono cloni se uno è un duplicato dell'altro (più o meno esatto dal punto di vista lessicale e sintattico), o se presentano una similarità semantica in quanto implementano una stessa funzione (hanno pre- e post-condizioni simili). Nonostante i linguaggi di programmazione offrono vari meccanismi di astrazione per facilitare l'incapsulazione e riutilizzo di codice, il metodo del "copia-ed-incolla" ("copy-and-paste"), è ancora molto usato per copiare e incollare un segmento di codice in un'altra posizione dello stesso programma o in un altro modulo per replicare lo stesso comportamento in più punti del sistema sw. Ciò causa la coesistenza di più copie di segmenti di codice esatti o molto simili nel sistema. Spesso questa operazione di copia è accompagnata da leggere modifiche nel codice clonato come la ridenominazione delle variabili o l'inserimento/eliminazione di parte del codice, per meglio adattarlo al nuovo contesto.

Vari sono i motivi per cui si effettua la duplicazione/clonazione di codice.

Tra questi:

- uno stretto e rigido vincolo temporale, o un'urgenza, che spinge gli utilizzatori a riusare, copiandolo, codice che implementa un comportamento simile a quello richiesto, rimandando a tempi successivi una migliore soluzione progettuale;
- il riuso di codice affidabile, per evitare il rischio di introdurre difetti con la

produzione di codice nuovo

- la mancanza nel linguaggio di programmazione di costrutti che favoriscono e permettono l'incapsulazione e riuso del codice stile e modello mentale dello sviluppatore che lo porta, più o meno inconsapevolmente, a definire ed usare stessi/simili pattern di programmazione.

La figura 1.1 riassume i vari motivi che portano alla clonazione di codice

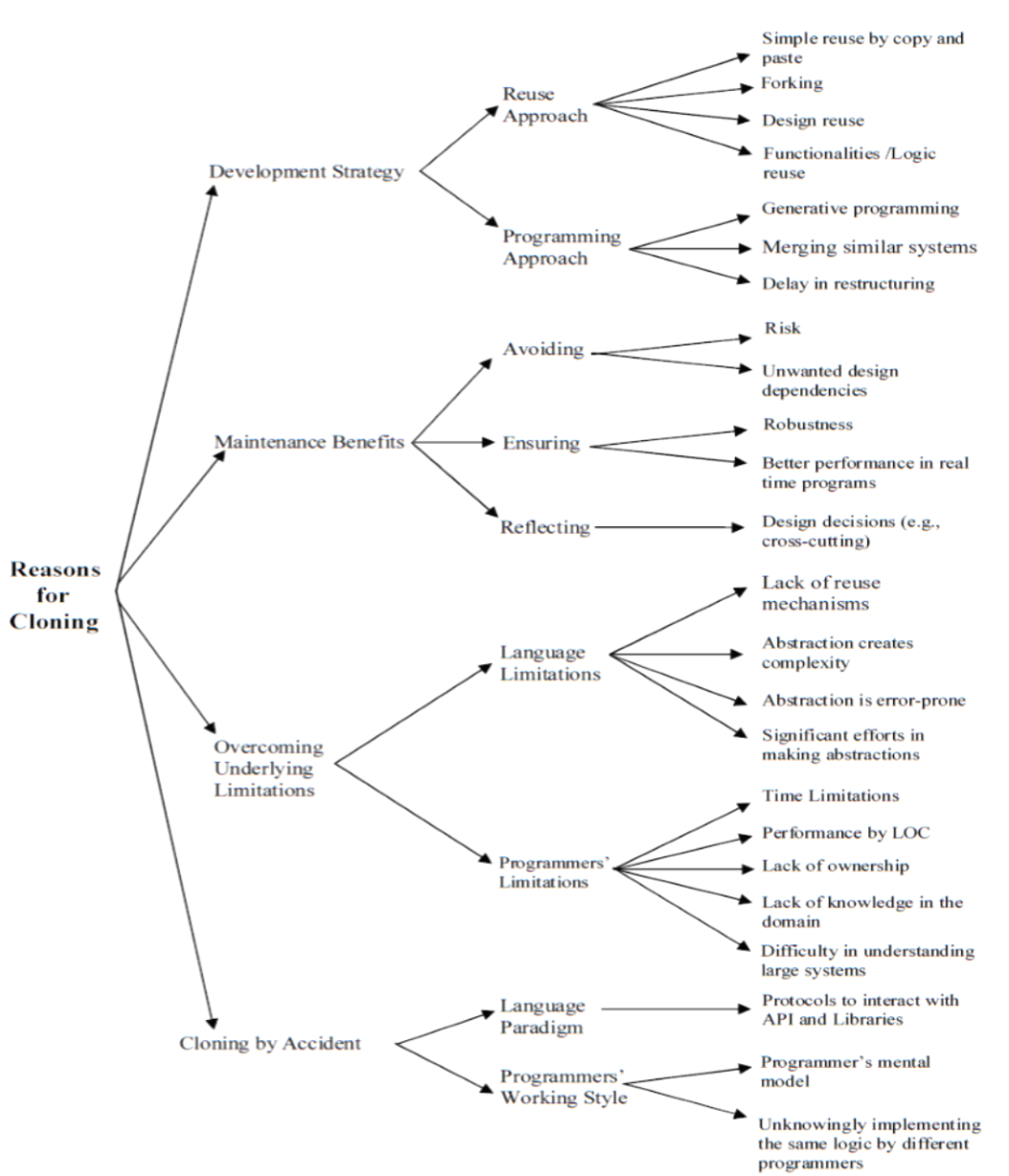


Figura 1.1 Principali Motivi della clonazione del codice

La *clonazione* può presentare alcuni vantaggi come il risparmio di tempo di sviluppo e aumento della percentuale di riuso del codice, ma essa è in considerata dannosa per la qualità specie la presenza di cloni può rendere più oneroso il testing, la manutenzione ed evoluzione del software,

Si pensi, ad esempio, al caso in cui è rilevato un errore in un segmento di codice clonato più volte nel sistema: in tal caso la stessa modifica per rimuovere l'errore deve essere applicata a tutti i segmenti clonati. Ciò richiede la precisa conoscenza dell'esistenza dei cloni e della loro localizzazione nel sistema, e comunque comporta un maggior onere della necessaria modifica (che va, appunto, ripetuta per ciascun clone).

1.2 Tipi di clone

In base al modo con cui un segmento di codice può essere clonato sono state definite, in letteratura, quattro tipi di cloni software [Koske] [Kaur]:

1. *Cloni esatti (exact clones)*: frammenti di codice identici eccetto per variazioni di spazi, commenti e layout; la figura 1.2 riporta un esempio di cloni esatti (tipo 1); in essi troviamo la stessa sequenza di istruzioni con stessi nomi delle variabili e con differenze relative ad aggiunta/eliminazione di commenti/spazi, o differente indentazione delle istruzioni.

```
int add(int num[],int x){
    int a=0;//add
     for(int m=0;m<x;m++){
     a=a+num[m];
     }
     return a;
     }
```

```
int add(int num[], int x){
int a=0;
for(int m=0;m<x;m++){
a=a+num[m];
}
return a;
}
```

```
int add(int num[],int x){
int a=0;//add
for(int m=0;m<x;m++){
a=a+num[m];
}
return a;
}
```

Figura 1.2 – Esempio di cloni esatti (tipo 1)

2. Cloni rinominati o parametrizzati (renamed clones): frammenti di codice strutturalmente/sintatticamente identici eccetto per variazioni dei nomi degli identificatori, letterali, tipi, per gli spazi bianchi, il layout e commenti. La figura 1.3 mostra un esempio di cloni di tipo 2; in essi troviamo la stessa sequenza di istruzioni ma i nomi di identificatori e/o variabili sono differenti e con differenze relative ad aggiunta/eliminazione di commenti/spazi, o differente indentazione delle istruzioni, mentre il comportamento è lo stesso.

```
int add(int num[],
int x){
    int a=0;//add
    for(int
m=0;m<x;m++){
    a=a+num[m];
    }
    return a;
    }
```

```
Int doadd(int no[],
int x){
int a=0;
for(int
m=0;m<x;m++){
add=add+no[m];
}
return add;
}
```

```
int addition(int s[],
int x){
int a=0;//add
for(int
m=0;m<x;m++){
a=a+s[m];
}
return a;
}
```

Figura 1.3 – Esempio di cloni di tipo 2

3. Near-Miss Clones: frammenti di codice simili tra loro ma con modifiche come l'aggiunta o rimozione di istruzioni, identificatori e/o modifiche come nei tipi 1 o 2 . La figura 1.4 riporta un esempio di cloni di tipo 3

```
int addition (int
num[], int n){
 int sum=0;//sum
 for(int
i=0;i<n;i++){
 sum=sum+num[i];
 }
 return sum;
 }
```

```
int doadd(int
no[], int n){
int s=0;
for(int
i=0;i<n;i++){
s+=no[i];
}
return s;
}
```

```
int sum(int a[],int
n){
int p=0;//sum
for(int i=0;i<n;){
p=p+a[i];
i++;
}
return p;
}
```

Figura 1.4 – Esempio di cloni di tipo 3

4. Cloni semantici (tipo 4): frammenti di codice che eseguono la stessa elaborazione (hanno lo stesso comportamento funzionale) ma implementati con differenti varianti sintattiche. La figura 1.5 riporta un esempio di cloni di tipo 4

```
int add(int no[],int n){
int sum=0;
for(int p=0;p<n;p++){
sum=sum+no[i];
}
return sum;
}
```

```
int add(int no[],int n){
if(n==1)
return no[n-1];
else
return no[n-1]+add[no,n-1];
}
```

Figura 1.5 - Esempio di cloni di tipo 4

La maggior parte degli studi per il rilevamento di cloni in un sistema software sono relativi all'individuazione di cloni di tipo 1, 2 e 3.

1.3 Tecniche di individuazione cloni

L'operazione di "rilevazione dei cloni" può essere conseguita attraverso diverse tecniche di classificazione, successivamente riportate:

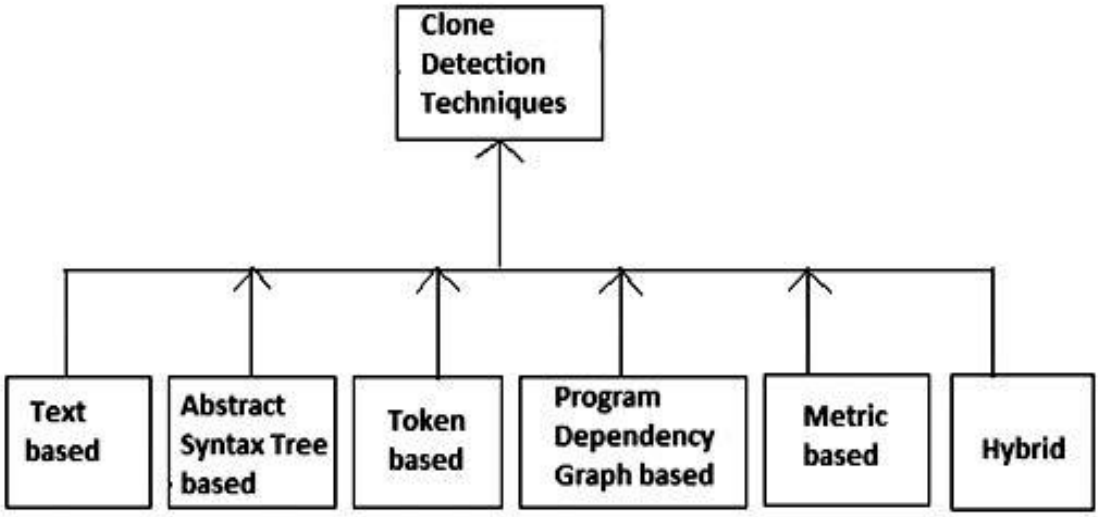


Figura 1.6 – Tecniche di classificazione cloni

- *Text based*: la rilevazione viene eseguita in base alla somiglianza testuale piuttosto che sulla base della somiglianza sintattica e semantica

- *Abstract Syntax Tree based*: i cloni sono rilevati comparando gli alberi astratti della sintassi che rappresentano i frammenti da confrontare, è quindi effettuato un confronto basato sulla struttura sintattica del codice.

- *Token based*: è effettuata un'analisi lessicale del codice che è convertito in una sequenza di token. La sequenza di token è analizzata per individuare sottosequenze uguali che corrispondono a frammenti. Cloni esatti e cloni sintattici sono tipicamente individuati con questa tecnica.

- *Programm Dependency Graph based*: partendo dal codice sorgente viene realizzato il grafo delle dipendenze del programma che include il controllo del flusso e il flusso dei dati. Il grafo viene analizzato per individuare sottografi simili che corrispondono a frammenti clonati

- *Metric based*: sono calcolate significative misure del codice che tengono conto di vari aspetti circa la struttura del codice e dei dati, i nomi di variabili, i letterali. Le parti del codice che evidenziano specificate caratteristiche metriche comparabili sono considerate come cloni.

- *Hybrid*: combinano due o più tecniche prima citate per rilevare i cloni; questa tecnica fornisce un risultato migliore rispetto alle tecniche normali. Ad esempio: tecniche basate su grafi e su metriche possono essere utilizzata in combinazione per ottenere risultati più precisi.

1.4 Processo di individuazione clone

La figura 1.7 rappresenta un tipico processo per la identificazione di cloni in un sistema.

Questo processo è piuttosto costoso e richiede una buona velocità di calcolo. I cloni vengono rilevati in base alla somiglianza.

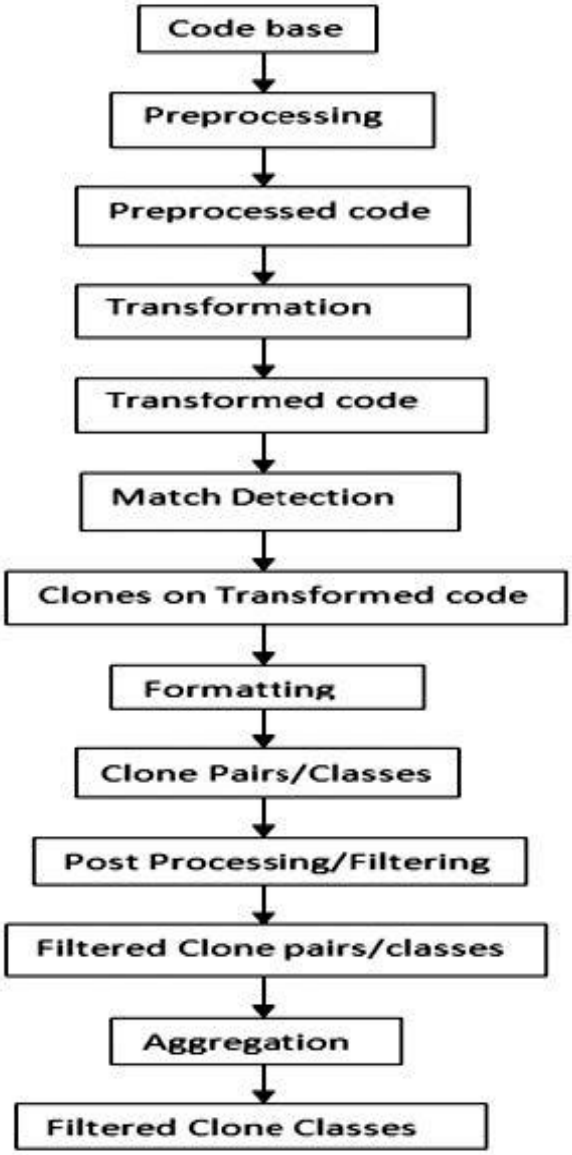


Figura 1.7 – Processo identificazione cloni

Preprocessing: questa fase prevede due passaggi: nel primo, conosciuto come segmentazione, viene suddiviso il codice sorgente in sezioni, nel secondo vengono individuate le aree di confronto. Questa fase comprende:

1) Eliminazione di parti indesiderate: il codice sorgente è segmentato e le parti non di interesse vengono rimosse, il che può generare falsi valori positivi.

2) Definire le unità sorgenti: una volta completata la rimozione del codice indesiderato, il resto del codice sorgente è partizionato in modo tale da poter ottenere una porzione comune. Per esempio: in un programma; file, classi, funzioni / metodi, blocchi di inizio o fine o sequenza di linee sorgente.

3) Definire le unità da confrontare: segmentazione delle unità sorgenti per ottenere ulteriori unità più piccole per il confronto.

Trasformation: in questa fase le unità del codice sorgente vengono convertite in particolari rappresentazioni intermedie. Questo passaggio è ulteriormente suddiviso in:

1) Estrazione: per rendere il codice sorgente appropriato come input per l'algoritmo reale viene effettuata la conversione

2) Tokenizzazione: ogni riga del codice sorgente è isolata in token.

3) Parsing: per individuare i cloni nell'approccio sintattico, viene utilizzato un albero astratto della sintassi per il confrontare sottoalberi. È anche possibile utilizzare l'approccio basato su metrica.

Match detection: il codice trasformato ottenuto dai passaggi precedenti viene immesso in un algoritmo di confronto in cui tutte le unità trasformate da confrontare vengono valutate sulla base alla loro similarità per determinare eventuali corrispondenze. L’algoritmo produce un insieme di coppie di candidati cloni. Gli algoritmi utilizzati in questa fase sono: suffix tree dynamic pattern matching e hash esteem examination.

Formatting: l'elenco di coppie di cloni ottenuto dall'algoritmo di confronto viene trasformato nel relativo elenco di coppie di cloni del codice originale.

Post processing/filtering: questo passaggio è ulteriormente suddiviso in due parti:

1) analisi manuale: qui i falsi positivi vengono filtrati da esperti umani.

2) euristica automatizzata: alcuni parametri sono già impostati in base agli scopi di filtraggio. Per esempio: lunghezza, frequenza, diversità ecc.

Aggregation: per ridurre la quantità di informazioni e/o facilitarne l'analisi, i cloni possono essere raccolti in classi di cloni per eseguire successivi esami o misurazioni.

Capitolo 2: Il tool di Clone Detection NiCad

2.1 Principali caratteristiche di NiCad

NiCad [Nicad] è uno strumento scalabile e flessibile di rilevamento di cloni, utilizzabile da riga di comando, che riceve come input una directory sorgente, in cui sono contenuti i file sorgente da esaminare, e fornisce come output un file .XML e un file .HTML in cui sono riportate le informazioni relative ai cloni trovati.

Il processo di rilevamento di NiCad prevede tre fasi principali: l'analisi, la normalizzazione e il confronto.

Nella prima fase vengono analizzati i file sorgenti per estrarre tutti i frammenti di codice secondo una predefinita granularità, come ad esempio funzioni o blocchi dove ciascun frammento estratto è un "potenziale clone".

Nella seconda fase, i frammenti di codice estratti possono essere normalizzati, filtrati o astratti prima del confronto.

Nella fase di confronto, i frammenti sono confrontati a livello lineare usando un algoritmo LCS (Longest Common Subsequence) per rilevare frammenti di codice simili.

Il confronto è parametrizzato rispetto ad un valore soglia che consente il rilevamento di cloni near-miss. Per esempio, una soglia di 0.0 rileva solo cloni esatti, 0.1 rileva quelli che possono differire fino al 10% di linee normalizzate, 0,2 fino al 20%, e così via.

A differenza di molti altri programmi, NiCad crea direttamente classi di cloni che contengono cloni dello stesso tipo che si riferiscono allo stesso frammento di codice in file sorgenti diversi.

2.2 Applicazione di NiCad

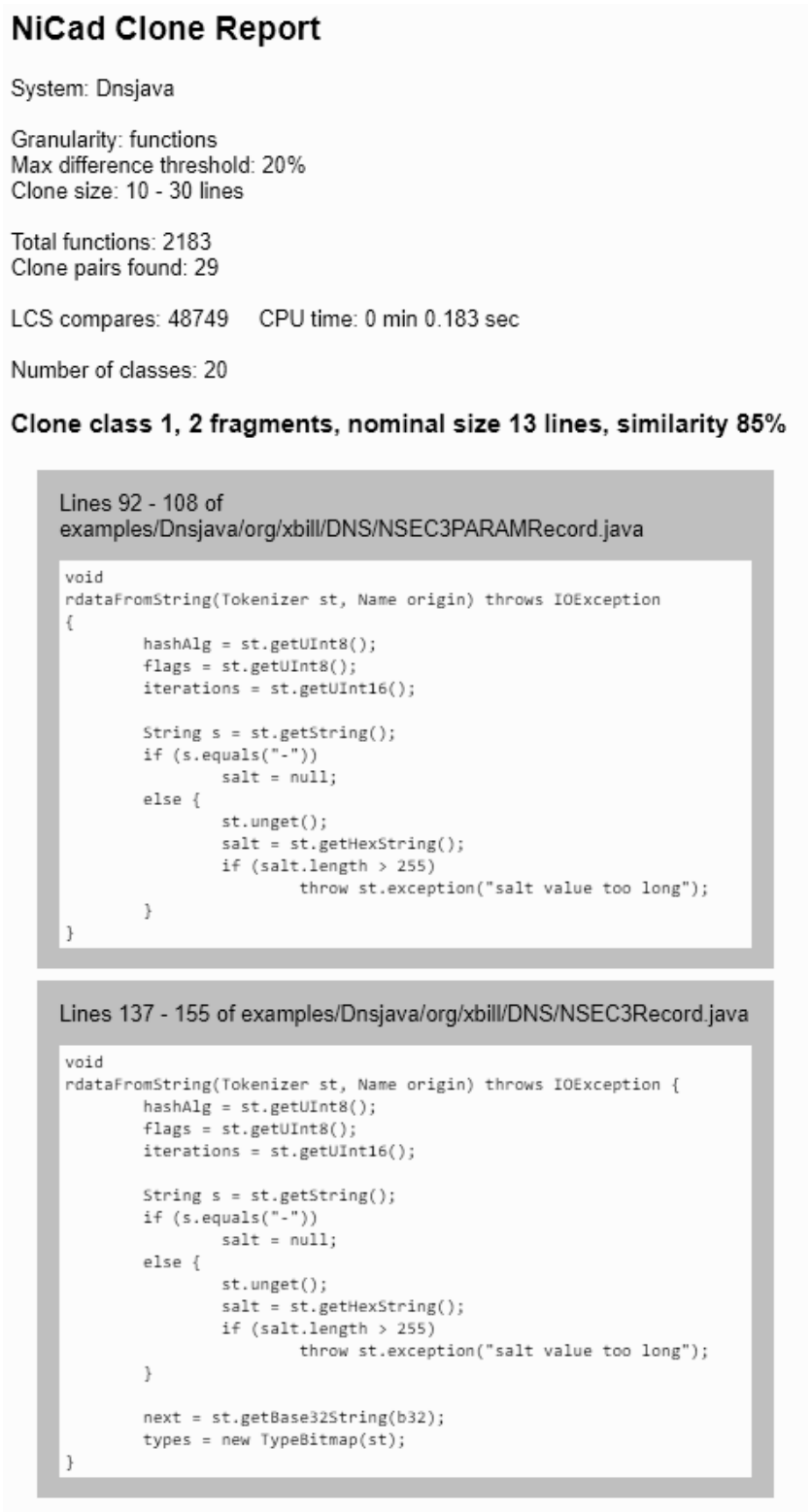
NiCad è stato utilizzato per rilevare i cloni di due sistemi: Dnsjva e Tika. NiCad è stato avviato dal terminale con il comando “nicad4 functions java” seguito dal percorso della cartella contenente i file sorgenti.

La figura 2.1 riporta un esempio del report generato da NiCad circa l’analisi dei cloni in un sistema software analizzato (il sistema Dnsjava) con la soglia pari al 20%, la granualità uguale a quella di una funzione, linguaggio java e intervallo di linee consecutive da 10 a 30. Il report indica che sono state analizzate 2183 funzioni e trovate 29 coppie di cloni.

```
NiCad Clone Detector v4.0 (15.2.16)

config=./config/default.cfg
system=examples/Dnsjava
threshold=0.20
granularity=functions
language=java
transform=none
rename=none
filter=none
abstract=none
normalize=none
cluster=yes
report=yes
include=
exclude=

gio 31 mag 2018, 11.59.53, CEST

Using previously extracted functions from java files in examples/Dnsjava

Extracted 2183 functions

Finding clones between 10 and 30 lines at UPI threshold 0.20

real    0m0.271s
user    0m0.122s
sys     0m0.087s

Found 29 clone pairs
```

Figura 2.1 – Analisi cloni di Dnsjava

La figura 2.2 riporta un esempio del report relativo a due cloni individuati da NiCad nel sistema Dnsjava indicando anche che essi sono formati da 13 linee e con una similarità del 85%. Sono anche riportate i due frammenti formanti la coppia di cloni individuati.



NiCad Clone Report

System: Dnsjava

Granularity: functions
Max difference threshold: 20%
Clone size: 10 - 30 lines

Total functions: 2183
Clone pairs found: 29

LCS compares: 48749 CPU time: 0 min 0.183 sec

Number of classes: 20

Clone class 1, 2 fragments, nominal size 13 lines, similarity 85%

Lines 92 - 108 of examples/Dnsjava/org/xbill/DNS/NSEC3PARAMRecord.java

```
void
rdataFromString(Tokenizer st, Name origin) throws IOException
{
        hashAlg = st.getUInt8();
        flags = st.getUInt8();
        iterations = st.getUInt16();

        String s = st.getString();
        if (s.equals("-"))
                salt = null;
        else {
                st.unget();
                salt = st.getHexString();
                if (salt.length > 255)
                        throw st.exception("salt value too long");
        }
}
```

Lines 137 - 155 of examples/Dnsjava/org/xbill/DNS/NSEC3Record.java

```
void
rdataFromString(Tokenizer st, Name origin) throws IOException {
        hashAlg = st.getUInt8();
        flags = st.getUInt8();
        iterations = st.getUInt16();

        String s = st.getString();
        if (s.equals("-"))
                salt = null;
        else {
                st.unget();
                salt = st.getHexString();
                if (salt.length > 255)
                        throw st.exception("salt value too long");
        }

        next = st.getBase32String(b32);
        types = new TypeBitmap(st);
}
```

Figura 2.2 – Esempio di cloni in Dnsjava

Il tool NiCad produce anche report in formato .xml indicanti informazioni circa i cloni identificati. Nella Tabella 2.1 è riportato un estratto da tale file .xml, relativa all'analisi del sistema Dnsjava, con la descrizione delle varie parti

Il codice in formato .xml riportato in seguito è un esempio di output di più cloni analizzati in Dnsjava suddivisi in classi di cloni caratterizzate dal numero di linee dei cloni e dalla loro similarità.

<class classid="1" nclones="2" nlines="22" similarity="77"> <source file="examples/Dnsjava/org/xbill/DNS/Zone.java" startline="305" endline="328" pcid="136"></source>
• class classid: indica la classe del clone, ovvero l dei frammenti di codice cloni trovati • nclones: il numero di cloni costituenti quella classe • nlines: il numero di linee di codice formanti il clone • similarity: la percentuale di similarità dei cloni nella classe • source file: nome del file sorgente in cui è stato rilevato il clone • startline: numero di linea di codice da dove inizia il clone nel file sorgente • endline: numero di linea di codice da dove inizia il clone nel file sorgente • pcid: codice identificatore del clone

Tabella 2.1: Estratto del file xml prodotto da NiCad e relativa legenda

Capitolo 3: Il sistema di controllo versione Git e di hosting GitHub

3.1 Principali caratteristiche di Git

Git [Git] è un software di controllo versione distribuito utilizzabile da interfaccia a riga di comando, creato da Linus Torvalds nel 2005. La sua progettazione si ispirò a strumenti (allora proprietari) analoghi come BitKeeper e Monotone.

Git nacque per essere uno strumento per facilitare lo sviluppo del kernel Linux ed è diventato uno degli strumenti di controllo versione più diffusi.

Il progetto di Git è la sintesi dell'esperienza di Torvalds nel mantenere un grande progetto di sviluppo distribuito, della sua intima conoscenza delle prestazioni del file system, e di un bisogno urgente di produrre un sistema soddisfacente in poco tempo. Queste influenze hanno condotto alle seguenti scelte implementative:

- *Forte supporto allo sviluppo non lineare*: Git supporta diramazione e fusione (branching and merging) rapide e comode, e comprende strumenti specifici per visualizzare e navigare una cronologia di sviluppo non lineare. Un'assunzione centrale in Git è che una modifica verrà fusa più spesso di quanto sia scritta, dato che viene passata per le mani di vari revisori.

- *Sviluppo distribuito*: Git dà a ogni sviluppatore una copia locale dell'intera cronologia di sviluppo, e le modifiche vengono copiate da un tale repository a un altro. Queste modifiche vengono importate come diramazioni aggiuntive di sviluppo, e possono essere fuse allo stesso modo di una diramazione sviluppata localmente.

- *Gestione efficiente di grandi progetti*: Git è molto veloce e scalabile. È tipicamente un ordine di grandezza più veloce degli altri sistemi di controllo

versione, e due ordini di grandezza più veloce per alcune operazioni.

- *Autenticazione crittografica della cronologia*: la cronologia di Git viene memorizzata in modo tale che il nome di una revisione particolare (secondo la terminologia Git, una "commit") dipende dalla completa cronologia di sviluppo che conduce a tale commit. Una volta che è stata pubblicata, non è più possibile cambiare le vecchie versioni senza che ciò venga notato.

- *Progettazione del toolkit*: Git è un insieme di programmi di base scritti in linguaggio C e molti script di shell che forniscono comodi incapsulamenti. È facile concatenare i componenti per fare altre cose utili.

- *Strategie di fusione intercambiabili*: come parte della progettazione del suo toolkit, Git ha un modello ben definito di una fusione incompleta, e ha più algoritmi per tentare di completarla. Se tutti gli algoritmi falliscono, tale fallimento viene comunicato all'utente e viene sollecitata una fusione manuale. Pertanto, è facile sperimentare con nuovi algoritmi di fusione.

3.2 Implementazione

Le primitive di Git non costituiscono inerentemente a un sistema di controllo della versione. Per esempio, Git non fornisce una numerazione progressiva delle revisioni del software.

Git ha due strutture dati, un *indice* modificabile che mantiene le informazioni sul contenuto della prossima revisione, e un *database di oggetti* a cui si può solo aggiungere e che contiene quattro tipi di oggetti:

- Un oggetto *blob* è il contenuto di un file. Gli oggetti blob non hanno nome, data, ora, né altri metadati. Git memorizza ogni revisione di un file come un oggetto blob distinto.
- Un oggetto *albero* è l'equivalente di una directory: contiene una lista di nomi di file, ognuno con alcuni bit di tipo e il nome di un oggetto blob o albero che è il file, il link simbolico, o il contenuto di directory. Questo oggetto descrive un'istantanea dell'albero dei sorgenti.
- Un oggetto *commit* (revisione) collega gli oggetti albero in una cronologia. Contiene il nome di un oggetto albero (della directory dei sorgenti di livello più alto), data e ora, un messaggio di archiviazione (log message), e i nomi di zero o più oggetti di commit genitori. Le relazioni tra i blob si possono trovare esaminando gli oggetti albero e gli oggetti commit.
- Un oggetto *tag* (etichetta) è un contenitore che contiene riferimenti a un altro oggetto, può tenere metadati aggiuntivi riferiti a un altro oggetto. Il suo uso più comune è memorizzare una firma digitale di un oggetto commit corrispondente a un particolare rilascio dei dati gestiti da Git.

Ogni oggetto è identificato da un codice hash SHA-1 del suo contenuto. Git calcola tale codice hash, e usa questo codice come nome dell'oggetto.

L'*indice* è uno strato intermedio che serve da punto di collegamento fra il database di oggetti e l'albero di lavoro.

Il database ha una struttura semplice. L'oggetto viene messo in una directory che corrisponde ai primi due caratteri del suo codice hash; Il resto del codice costituisce il nome del file che contiene tale oggetto. Quando si aggiunge un nuovo oggetto, questo viene memorizzato per intero dopo averlo compresso con zlib.

Questo fatto può far sì che in poco tempo venga occupato molto spazio sul disco fisso, perciò gli oggetti possono essere combinati in **pack**, che usano la compressione delta (memorizzando solo le modifiche tra un blob e un altro blob) per risparmiare spazio.

3.3 GitHub

GitHub [GitHub] è un servizio di hosting per progetti software. Il nome "GitHub" deriva dal fatto che GitHub è una implementazione dello strumento di controllo versione distribuito Git. Il sito è principalmente utilizzato dagli sviluppatori, che caricano il codice sorgente dei loro programmi e lo rendono scaricabile dagli utenti. Questi ultimi possono interagire con lo sviluppatore tramite un sistema di issue tracking, pull request e commenti che permette di migliorare il codice della repository risolvendo bug o aggiungendo funzionalità. Inoltre, GitHub elabora dettagliate pagine che riassumono come gli sviluppatori lavorano sulle varie versioni dei repository.

Nel caso di questo lavoro di tesi, GitHub è stato usato per accedere e scaricare i sistemi utilizzati nell'esperimento per associare i cloni software presenti in essi ai committers. In particolare, i sistemi recuperati da GitHub, sono:

- Dnsjava, è un'implementazione del DNS in java, può essere utilizzata per query, trasferimenti di zona e aggiornamenti dinamici
- NoteManager, è un applicazione che consente di gestire un insieme di note condivise, scritta in java per dispositivi Android.
- Tika, è un toolkit per il rilevamento e l'estrazione di metadati e il contenuto di testi strutturati in diversi documenti utilizzando le librerie di parser esistenti.

Capitolo 4: Il tool Sourcetree

4.1 Principali caratteristiche di Sourcetree

Le informazioni relative ai commit e i committers sono state recuperate utilizzando Sourcetree [Sourcetree], un client di Git gratuito che permette di interagire con i Git repository in modo più semplice ed efficace.

Esso permette di visualizzare e gestire i repository tramite la semplice GUI Git di Sourcetree oppure tramite terminale Git.

Nel caso specifico è stato utilizzato il comando git log tramite terminale Git che mostra i commits eseguiti nel repository in ordine cronologico inverso. In questo modo il commit più recente è il primo ad apparire. Ogni commit è elencato riportando il suo codice SHA-1, il nome e l'e-mail dell'autore (committer), la data di salvataggio e la descrizione del commit. Sono disponibili moltissime opzioni da passare al comando "git log" per vedere esattamente altre informazioni specifiche.

L'opzione utilizzata è "-p", che mostra le differenze introdotte da ciascuna commit.

Un'altra opzione interessante è "format", che permette di specificare la formattazione dell'output di log. Questa è specialmente utile quando si genera un output che sarà analizzo da una macchina, come nel caso specifico, perché quest'ultimo non cambierà con gli aggiornamenti di Git.

Di seguito alcuni esempi di formattazione dell'output:

Opzione	*Descrizione dell'output*
%H	Hash della commit
%h	Hash della commit abbreviato
%T	Hash dell'albero
%P	Hash del genitore
%an	Nome dell'autore
%ae	e-mail dell'autore
%ad	Data di commit dell'autore (il formato rispetta l'opzione --date=)
%ar	Data relativa di commit dell'autore
%cn	Nome di chi ha fatto la commit (committer, in inglese)
%ce	e-mail di chi ha fatto la commit
%cd	Data della commit
%s	Oggetto

E' stata infine aggiunta l'opzione "--date=iso" per ottenere come output la data nel formato *ISO 8601*, uno standard internazionale per la rappresentazione di date e orari, ovvero "yyy-MM-dd HH:mm:ss".

Il risultato è stato salvato in file di testo aggiungendo alla fine l'opzione ">" seguito dal nome del file.

Nello specifico è stata utilizzata la seguente istruzione:" git log –date=iso -p –pretty=format:"Commit:%n%H%n%cn%n%ce%n%cd%n%s" > nomeFile".

La figura 4.1 riporta un esempio di commit del progetto Dnsjava, in cui sono indicati:

- il codice identificatore del commit
- il nome e l'indirizzo di e-mail del committer
- la data in cui è stato effettuato il commit
- la descrizione/motivazione del commit
- il file sorgente oggetto del commit
- intervallo di modica all'interno del file

```
Commit:
849bde970e2349d828147453807898ed761cc2c7
bwelling
bwelling@c76caeb1-94fd-44dd-870f-0c9d92034fc1
2016-08-14 17:21:17 +0000
Fix typo.
diff --git a/org/xbill/DNS/OPENPGPKEYRecord.java b/org/xbill/DNS/OPENPGPKEYRecord.java
index 696eda7..359fb67 100644
--- a/org/xbill/DNS/OPENPGPKEYRecord.java
+++ b/org/xbill/DNS/OPENPGPKEYRecord.java
@@ -33,7 +33,7 @@ getObject() {
 public
 OPENPGPKEYRecord(Name name, int dclass, long ttl, byte [] cert)
 {
-   super(name, Type.CERT, dclass, ttl);
+   super(name, Type.OPENPGPKEY, dclass, ttl);
    this.cert = cert;
 }
```

Figura 4.1 – Esempio di report di un commit relativo al progetto Dnsjava

Capitolo 5: Il tool Clo_tter

5.1 Definizione caso studio

Deve essere realizzata una tool-chain di cui il tool Clo_tter è l'elemento finale. Lo scopo della tool-chain è di:

1. accedere alle informazioni di progetti open-source ospitati in GitHub, selezionare uno di tali progetti, scaricarne i file con il codice sorgente di una delle versioni del file e quelli relativi ai commit effettuati e memorizzare tali file in un repository locale;
2. analizzare il codice sorgente per l'individuazione in esso di cloni software;
3. analizzare il file con le informazioni relative ai commit per filtrare ed estrarre quelle di interesse;
4. analizzare le informazioni ottenute nei due punti precedenti in modo da riuscire ad associare ciascun clone al committer che ha effettuato un commit su quel frammento di codice clonato; tale committer è candidato ad essere considerato come il responsabile dell'introduzione del clone. Tali informazioni vanno registrate in una base di dati per permetterne successive ed ulteriori elaborazioni.

Per il punto 1) si usano le funzionalità e facility messe a disposizione da GitHub.
Per il punto 2) si utilizza il tool NiCad.
Per i punti 3) e 4) è sviluppato il tool Clo_tter, che permette ad un suo utente di selezionare uno dei progetti/sistemi software scaricati da GitHub e il cui codice sorgente è già stato analizzato per l'identificazione dei cloni in esso. Il tool Clo_tter effettua le elaborazioni relative al precedente punto 3. In particolare, per ogni commit, dal file scaricato da GitHub, sono estratte le informazioni relative al file sorgente oggetto del commit, l'identificativo del committer, il numero delle linee di codice

modificate dal commit. Tali informazioni sono confrontate con quelle relative a ciascun clone (in particolare sono confrontati i nomi dei file sorgenti ed i numeri di linea del clone all'interno del file sorgente) e quando si verifica un match (uguaglianza dei file sorgenti e numero linee modificate dal commit entro il range del numero di linee del frammento clone) il committer del commit verificante il match è associato a quel clone. È definito un apposito data base relazionale in cui sono memorizzate le informazioni prodotte da Clo_tter. In tal modo un utente può successivamente effettuare delle interrogazioni sul data base, ad esempio per scegliere tra i vari committers e visualizzare i cloni ad esso associati oppure, viceversa, scegliere tra i vari cloni e visualizzare i relativi commits e i committers. È possibile anche visualizzare tutti i commits, i cloni, i committers e le classi dei cloni relativi al progetto scelto.

La figurar 5.1 riporta l'architettura generale del progetto realizzato: recupero dei di progetto da GitHub, analisi del progetto tramite Sourcetree e NiCad per ottenere commits, committers e cloni, utilizzo di Clo_tter che con l'aiuto di un database realizza l'associazione clone-committer.

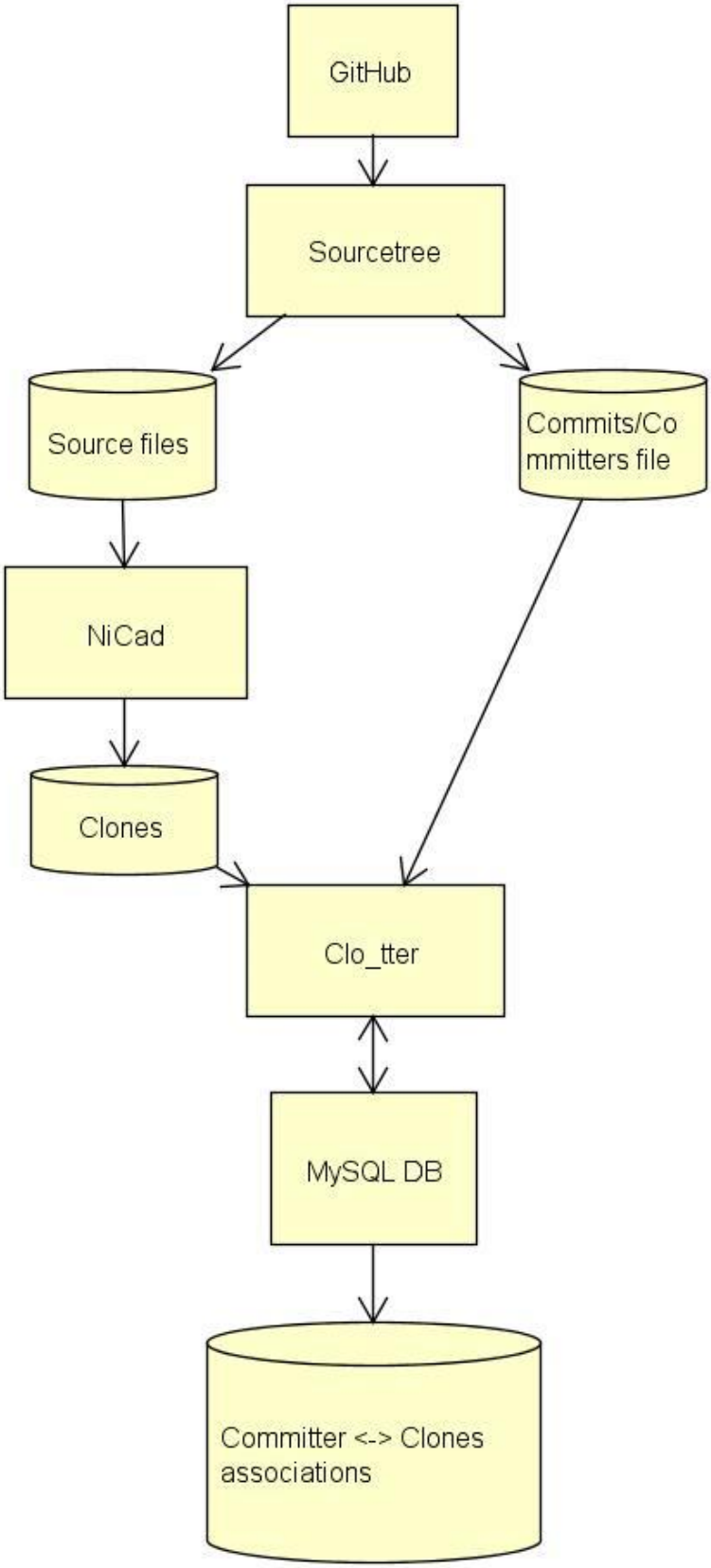


Figura 5.1 – Architettura generale

La figura 5.2 riporta un UML activity diagram modellante l'intero processo realizzato tramite la tool-chain descritta.

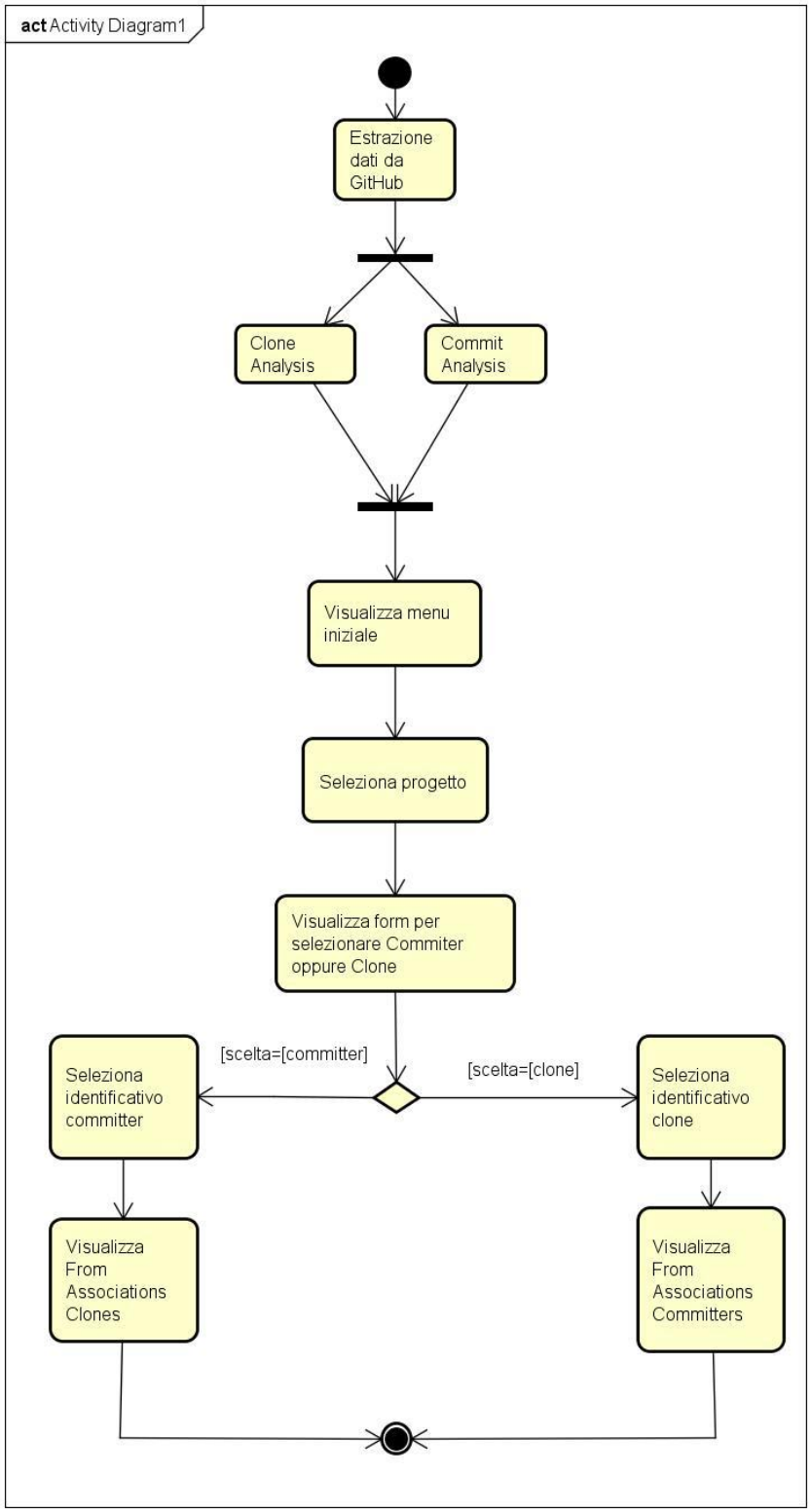


Figura 5.2 – Activity Diagram

5.2 Descrizione dei principali casi d'uso

Nome: Analisi_Dati_Progetto [UC1]
Attore: Ingegnere del software
Flusso eventi [popolamento base dati e produzione associazione cloni-committer]:
1. L'utente seleziona tra i vari progetti scaricati da GitHub e riportati in una lista.
2. Clo_tter effettua il parsing dei file con le informazioni relative ai commits e i committers e del file con le informazioni relative ai cloni e alle classi dei cloni, estrae da essi le informazioni di interesse, effettua le elaborazioni per definire le associazioni tra committers e cloni e salva i dati nel database.

Flusso evento alternativo A3[interruzione esecuzione Clo_tter]
*. Clo_tter smette di funzionare per problemi di elaborazione o relativi al collegamento con il database.

Pre-condizioni: sono disponibili i file dei cloni e dei commit per il progetto selezionato
Input: nome del progetto da analizzare
Condizioni finali: i file del progetto sono stati analizzati e le informazioni prodotte sono state salvate nel database
Output: tabelle del data base e le associazioni committers-cloni
Requisiti di qualità: le associazioni sono quelle corrette.

Nome: Visualizza_Committers_Cloni [UC2]

Attore: Ingegnere del software

Flusso eventi [visualizzazione cloni di un committer]:

1. È visualizzata la lista dei committers del progetto selezionato in precedenza

3. L'utente seleziona un committer

4. Clo_tter interroga la base di dati e visualizza tutti i cloni associati al committer selezionato

Flusso evento alternativo A1[visualizzazione committers si un clone]

1. È visualizzata la lista dei cloni del progetto selezionato in precedenza

2. L'attore seleziona un clone

4. Clo_tter interroga la base di dati e visualizza tutti i committers associati al clone selezionato

Flusso evento alternativo A3[interruzione esecuzione Clo_tter]

*. Clo_tter smette di funzionare per problemi di elaborazione o relativi al collegamento con il database.

Pre-condizioni iniziali: è stato selezionato un progetto e popolato la relativa base di dati, è stato eseguito UC1

Input: committer o clone di cui visualizzare le informazioni associate

Post-Condizioni: il contenuto della base di dati resta invariato

Output: lista associazioni committers-cloni

Requisiti di qualità: la lista associazioni committers-cloni è corretta

5.2.1 BOUNDARY USE CASE

Nel costruttore di **Controller** viene creata un'istanza **GestoreDB** alla quale vengono delegate tutte le operazioni utili a scrivere e leggere in memoria i dati persistenti.
In particolare, vengono scritte e lette le tabelle:

- Committers
- Commits
- Clones
- ClassClone
- Changes
- Ranges

5.3 Il design del tool Clo_tter

5.3.1 Design architetturale

Dovendo interagire con l'utente, l'architettura di Clo_tter è incentrata sul pattern **MVC**: (Model, View, Controller) che è adatto ad applicazioni interattive e fornisce un metodo efficiente per disaccoppiare la componente HCI con il resto dell'applicazione. In particolare, esso divide il sottosistema in tre componenti che gestiscono indipendentemente input, elaborazione, output:

- Model: rappresenta il modello dei dati di interesse per l'applicazione;
- View (GUI): fornisce una rappresentazione grafica del model;
- Controller: definisce la logica di controllo e le funzionalità applicative.

5.3.2 Design di Clo_tter

La figura 5.2 riporta lo UML Class Diagram di Clo_tter, riportante le principali classi che lo costituiscono e le loro relazioni.

Per motivi di spazio nella figura non sono riportati tutti i dettagli progettuali, quali quelli relativi ai Design Pattern utilizzati.

Uno è il **Singleton,** utilizzato per le classi di gestione ovvero Controller, GestoreCommits, GestoreClones, GestoreDB per avere una e una sola istanza di esse.

Un atro è il **Facade,** utilizzato nella classe GestoreDB insieme al package java.sql utilizzando un'interfaccia di più alto livello rendendo il sottosistema relativo all'accesso al database, più semplice.

Nel seguito è riportata una sintetica descrizione delle classi componenti la parte Model del pattern MVC il class diagram della figura 5.2:

- Commit: rappresenta i commit effettuati sul sistema software analizzato; è caratterizzata dai seguenti attributi: identificativo, data, descrizione e Committer;
- Committer: rappresenta i committer che hanno agito sul sistema software analizzato è caratterizzata dai seguenti attributi: nome ed email;
- Clone: rappresenta i cloni identificati nel sistema analizzati; è caratterizzata dai seguenti attributi: identificativo, file, prima linea del clone, ultima linea del clone, identificativo della classe del clone;
- Classe del clone: raggruppa tutte i frammenti di codice clone tra loro; è caratterizzata dai seguenti attributi: identificativo, numero di cloni, numero di linee e similarità.
- File: rappresenta un file sorgente del sistema analizzato, oggetto delle operazioni di change in un commit; è caratterizzata dal nome del file;
- Change: rappresenta le varie operazioni di modifica che sono eseguite in un commit; è caratterizzata da un identificativo, identificativo del relativo Commit e del file cui è applicato;
- Range: rappresenta l'intervallo delle linee di codice di un file sorgente cui è stato applicato un change in un commit; è caratterizzata da un identificativo, identificativo del relativo Change, la prima riga e l'intervallo di modifica

Un Commit può essere formato da diversi *changes* a più file sorgenti differenti, mentre il *change* può essere formato da più *ranges* riferiti sempre allo stesso file.
Inoltre, il class diagram riporta le classi costituenti la parte Controller dello MVC.
La classe GestoreDB ha la responsabilità di interfacciare l'applicazione con il database, permettendo di leggere o scrivere persistenti in memoria.
La classe Controller ha il compito di determinare il modo in cui l'applicazione risponde agli input dell'utente.

La classe GestoreCommits ha la responsabilità di leggere i commits, committers e files dal file generato da Sourcetree e di istanziare le relative liste.

La classe GestoreClones ha la responsabilità della lettura dei cloni e delle classi dei cloni e di istanziare le relative liste.

Il class diagram riporta anche un package GUI, che rappresenta la parte view dello MVC, raggruppante tutte le classi per realizzare l'interfaccia utente.

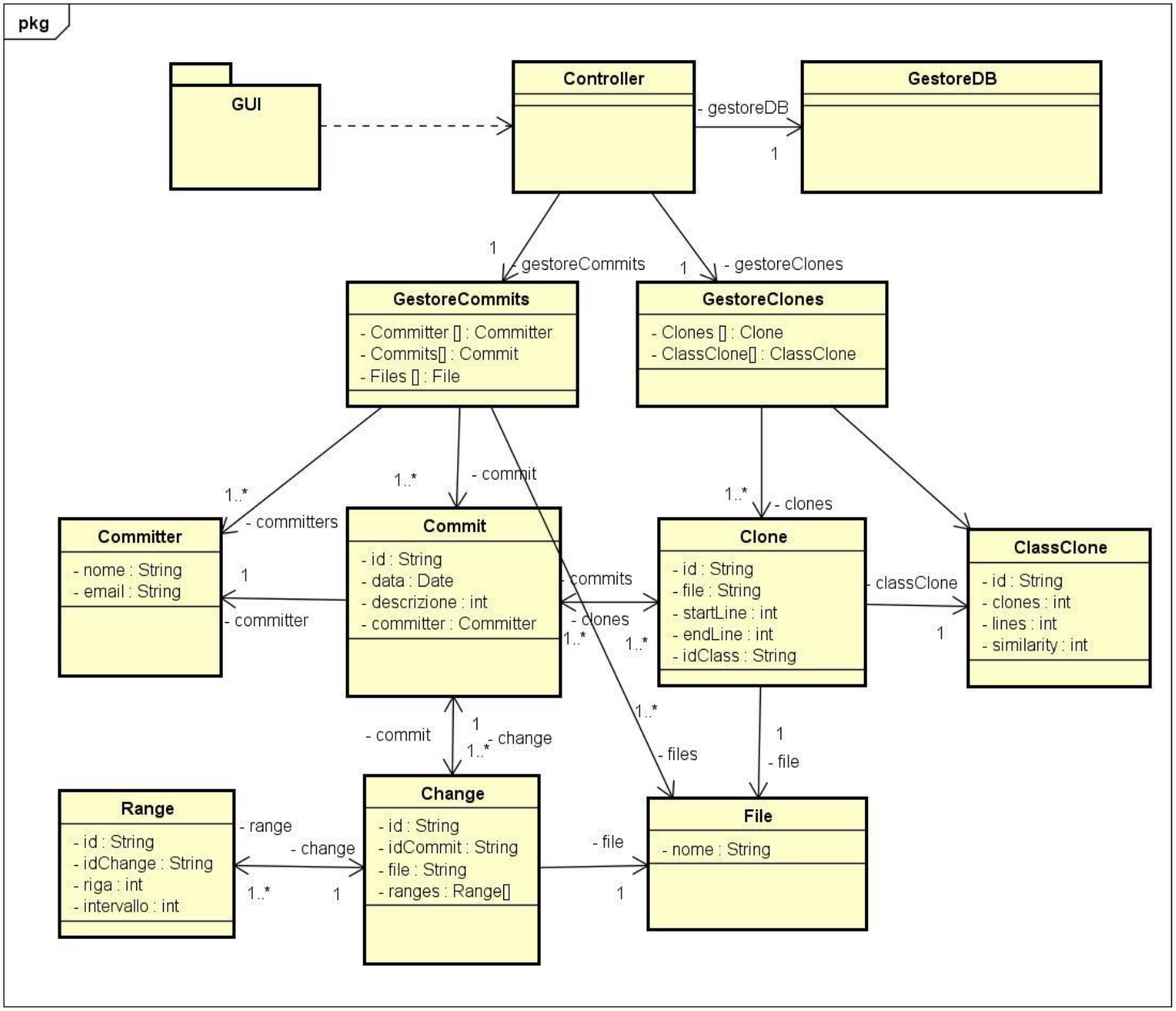


Figura 5.3 – Class Diagram

La figura 5.3 riporta le classi contenute nel package GUI, che modellano l'interfaccia grafica utente e che consentono all'utente di interagire con il tool.

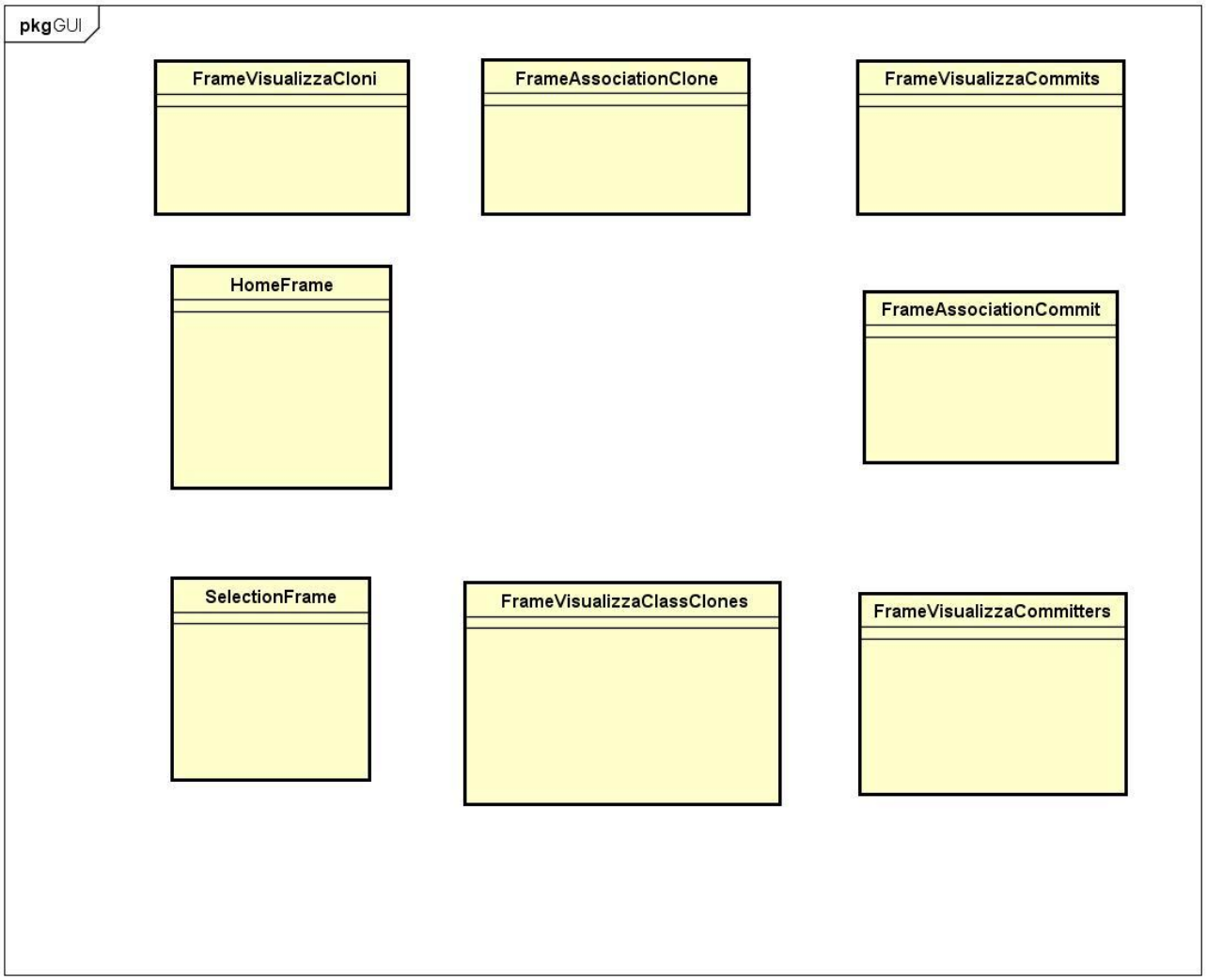


Figura 5.4 – Package Diagram GUI

Il diagramma riporta anche la associazioni tra le varie classi descritte sinteticamente nel seguito:

● GUI -> Controller: associazione con molteplicità 1 in quanto alla singola istanza della GUI è associata, come reference attribute, una singola istanza del Controller

● Controller -> GestoreDB: associazione con molteplicità 1 in quanto alla singola istanza del Controller è associata, come reference attribute, una singola istanza del GestoreDB

● Controller-> GestoreCommits: associazione con molteplicità 1 in quanto alla singola istanza del Controller è associata, come reference attribute, la singola istanza del GestoreCommits (singleton), denominata gestoreCommits.

● Controller-> GestoreClones: associazione con molteplicità 1 in quanto alla singola istanza del Controller è associata, come reference attribute, la singola istanza del GestoreClones (singleton), denominata gestoreClones.

● GestoreCommits-> Commit: associazione con molteplicità 1..* in quanto alla singola istanza del GestoreCommits è associata, come reference attribute, una collezione di oggetti della classe Commit, commits.

● GestoreCommits-> Committer: associazione con molteplicità 1..* in quanto alla singola istanza del GestoreCommits è associata, come reference attribute, una collezione di oggetti della classe Committer, committers.

● GestoreCommits-> File: associazione con molteplicità 1..* in quanto alla singola istanza del GestoreCommits è associata, come reference attribute, una collezione di oggetti della classe File, files.

● GestoreClones -> Clone: associazione con molteplicità 1..* in quanto alla singola istanza del GestoreClones è associata, come reference attribute, una collezione di oggetti della classe Clone, clones.

● GestoreClones -> ClassClone: associazione con molteplicità 1..* in quanto alla singola istanza del GestoreClones è associata, come reference attribute, una collezione di oggetti della classe ClassClone, classClone.

● Commit -> Committer: associazione con molteplicità 1 in quanto alla classe Commit è associata, come reference attribute, la classe Committer, denominata committer.
● Commit -> Change: associazione con molteplicità 1..* in quanto alla classe Commit è associata, come reference attribute, una collezione di oggetti della classe Change, changes.
● Change -> Commit: associazione con molteplicità 1 in quanto alla classe Change è associata, come reference attribute, la classe Commit, denominata commit.
● Change -> Range: associazione con molteplicità 1..* in quanto alla classe Change è associata, come reference attribute, una collezione di oggetti della classe range, ranges
● Range-> Change: associazione con molteplicità 1 in quanto alla classe Range è associata, come reference attribute, la classe Change, denominata change.
● Change -> File: associazione con molteplicità 1 in quanto alla classe Change è associata, come reference attribute, la classe File, denominata file.
● Clone -> ClassClone: associazione con molteplicità 1 in quanto alla classe Clone è associata, come reference attribute, la classe ClassClone, denominata classClone.
● Commit -> Clone: associazione con molteplicità 1..* in quanto alla classe Commit è associata, come reference attribute, una collezione di oggetti della classe Clone, clones.
● Clone -> Commit: associazione con molteplicità 1..* in quanto alla classe Clone è associata, come reference attribute, una collezione di oggetti della classe Commit, commits.

5.3.5 Sequence Diagram

La figura 5.5 riporta il sequence diagram relativo al primo use case che descrive l'elaborazione dei dati di un progetto da parte di Clo_tter e il loro salvataggio nel database dopo che l'utente ha scelto un progetto.

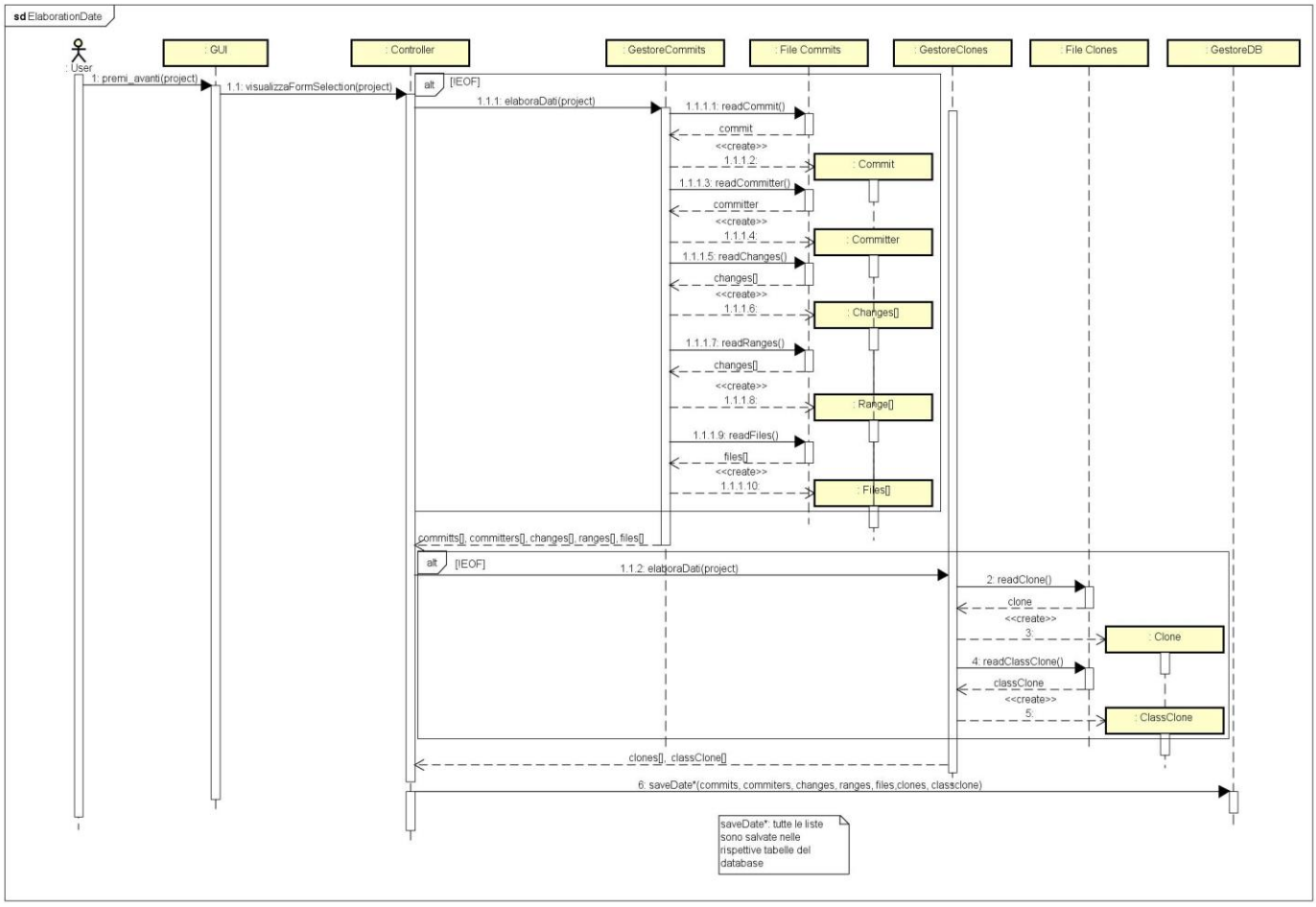


Figura 5.5 – Sequence Diagram relativo UC1

La figura 5.6 riporta il sequence diagram relativo al secondo use case che descrive l'interazione di un utente con Clo_tter in cui è richiesta la visualizzazione dei cloni associati a un committer o viceversa, la visualizzazione dei committer associati a un clone.

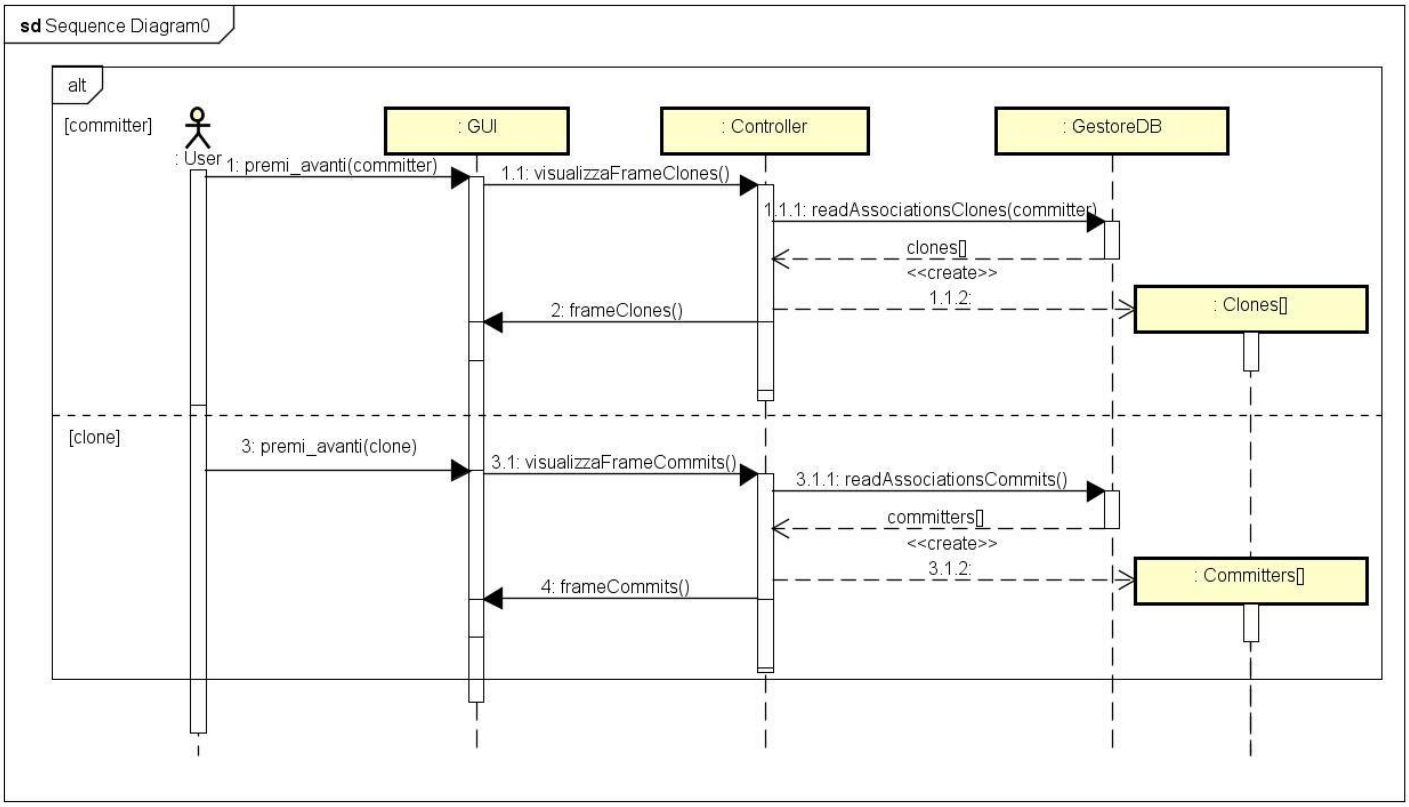


Figura 5.6 – Sequence Diagram relativo a UC2

5.4 Implementazione

5.4.3 Interfacciamento database - applicazione

Il database è gestito tramite il DBMS *MySQL*.
L'interfacciamento con l'applicativo Java è eseguito attraverso il connettore *JDBC.*
In particolare, è la Singleton class **GestoreDB** che provvede alla comunicazione con il database.

5.4.1 Component Diagram

La figura 5.7 riporta un component diagram modellante i principali componenti di Clo_tter: Clo_tter.management, relativo all'insieme delle classi di gestione; Clo_tter.baseClass, relativo all'insieme delle classi base e Clo_tter.gui,relativo al l'insieme delle classi realizzanti l'interfaccia utente.
Clo_tter può connettersi a un DBMS con l'utilizzo di JDBC API, un driver che consente l'accesso e la gestione della persistenza dei dati.

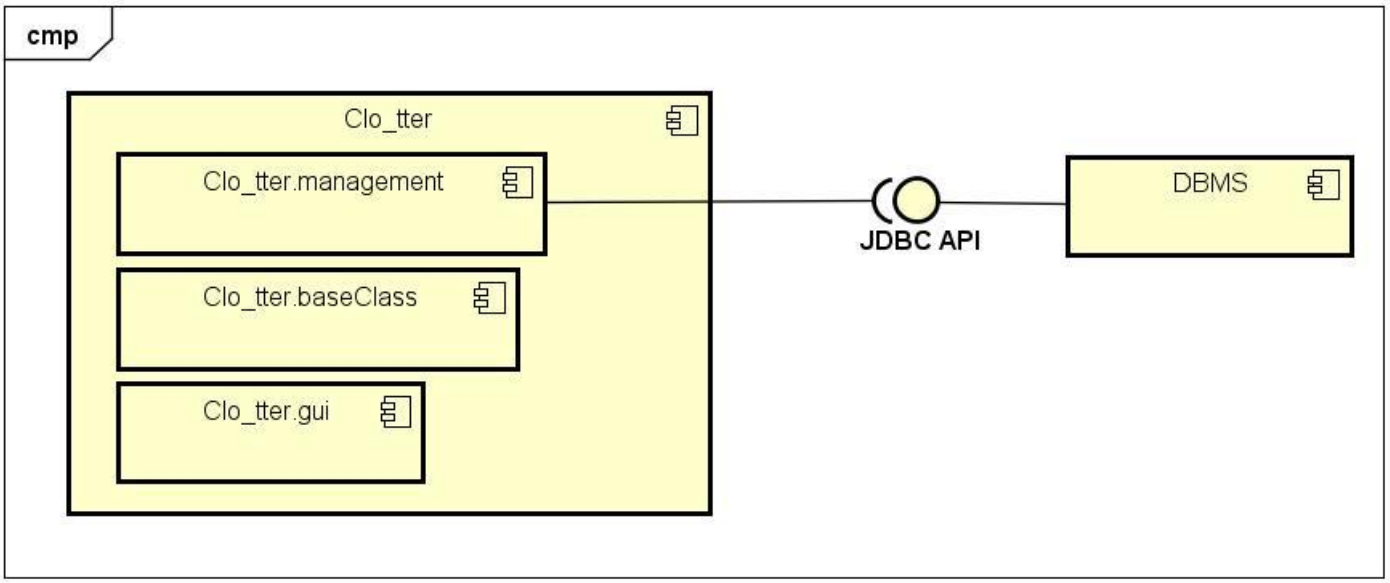


Figura 5.7 – Component Diagram

5.4.2 Schema logico DB

Nel seguito è riportato lo schema logico delle tabelle costituenti il data base di Clo_tter:

Commits= {id, emailCommitter, data, descrizione}

Committers= {email, nome}

Changes= {id, idCommit, file}

Ranges= {id, idChange, riga. intervallo}

Files={nomeFile}

Clones= {file, startLine, endLine, pcid, classid}

ClassClones= {id, clones, lines, similarity}

Associations= {idClone, idCommit}

La figura 5.8 riporta uno screen shot relativo al contenuto della tabella Clones del progetto Dnsjava.

	file	startline	endline	pcid	classid
	org/xbill/DNS/DSRecord.java	103	117	1304	2
	org/xbill/DNS/TLSARecord.java	110	122	1320	3
	org/xbill/DNS/SMIMEARecord.java	112	124	1352	3
	tests/org/xbill/DNS/SingleCompressedNameBas...	84	108	1478	4
	tests/org/xbill/DNS/MessageTest.java	48	65	1502	5
	tests/org/xbill/DNS/MessageTest.java	67	85	1503	5
	org/xbill/DNS/NSEC3PARAMRecord.java	92	108	153	1
	tests/org/xbill/DNS/HeaderTest.java	55	69	1533	6
	tests/org/xbill/DNS/HeaderTest.java	71	86	1534	6
	tests/org/xbill/DNS/DNSKEYRecordTest.java	45	57	1552	7
	tests/org/xbill/DNS/DNSKEYRecordTest.java	66	88	1554	8
	tests/org/xbill/DNS/KEYRecordTest.java	45	57	1572	7
	tests/org/xbill/DNS/KEYRecordTest.java	66	88	1574	8
	tests/org/xbill/DNS/SOARecordTest.java	369	380	1622	9
	tests/org/xbill/DNS/SOARecordTest.java	420	431	1625	9
	tests/org/xbill/DNS/DNAMERecordTest.java	49	61	1630	10
	tests/org/xbill/DNS/SingleNameBaseTest.java	139	163	1650	4
	tests/org/xbill/DNS/MFRecordTest.java	49	61	1668	11
	tests/org/xbill/DNS/SectionTest.java	74	80	1795	12
	tests/org/xbill/DNS/SectionTest.java	82	88	1796	12
	tests/org/xbill/DNS/SetResponseTest.java	127	148	1803	13
	tests/org/xbill/DNS/SetResponseTest.java	150	170	1804	13

Figura 5.8 – Esempio tabella dei cloni

5.4.4 GUI (Graphic User Interface)

Nel seguito sono riportate figure mostranti le principali interfacce utente.

Appena avviata l'applicazione, è presentata la finestra principale dalla quale è possibile usufruire delle principali funzionalità del sistema.

L'immagine 5.9 riporta l'interfaccia iniziale in cui è possibile scegliere tra i vari progetti.

Figura 5.9 – Interfaccia iniziale

Dopo aver selezionato un progetto Clo_tter elabora i relativi dati di input (i cloni ed i commit) popola il data base e quindi presenta un'interfaccia in cui è possibile selezionare un committer o un clone identificato nel progetto.

Nell'immagine 5.10 è riportata l'interfaccia in cui si può selezionare un committer tra quelli presentati nella lista di tutti i committer del sistema analizzato.

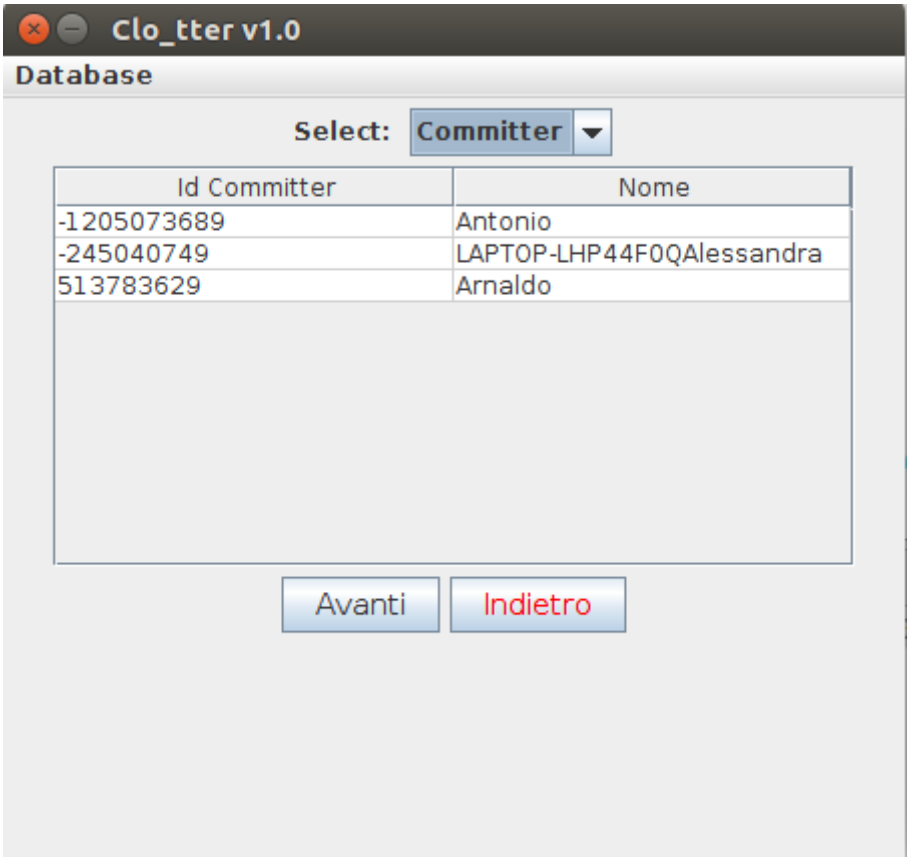


Figura 5.10 – Interfaccia di selezione committer

Analogamente l'immagine 5.11 è riportata l'interfaccia in cui si può selezionare un clone tra quelli presentati nella lista dei cloni identificati nel sistema.

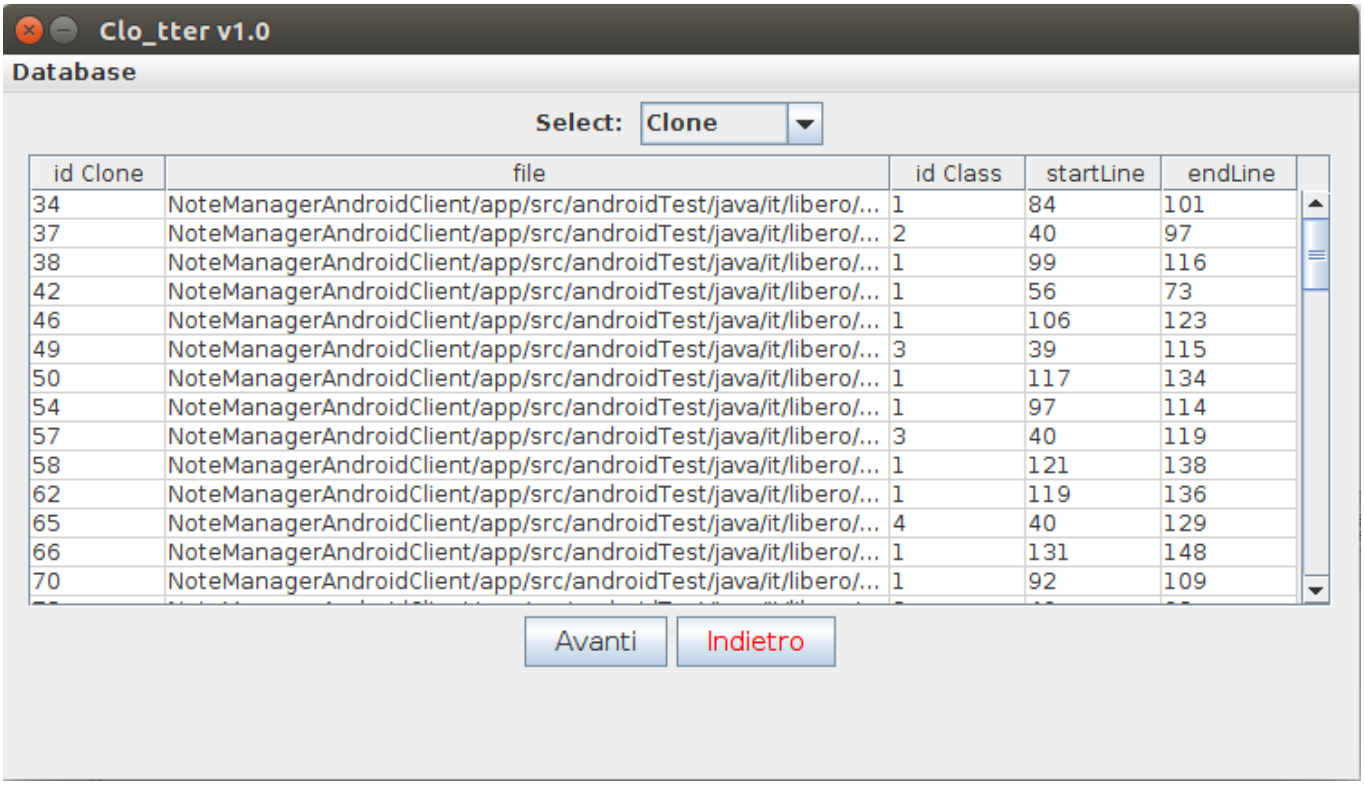
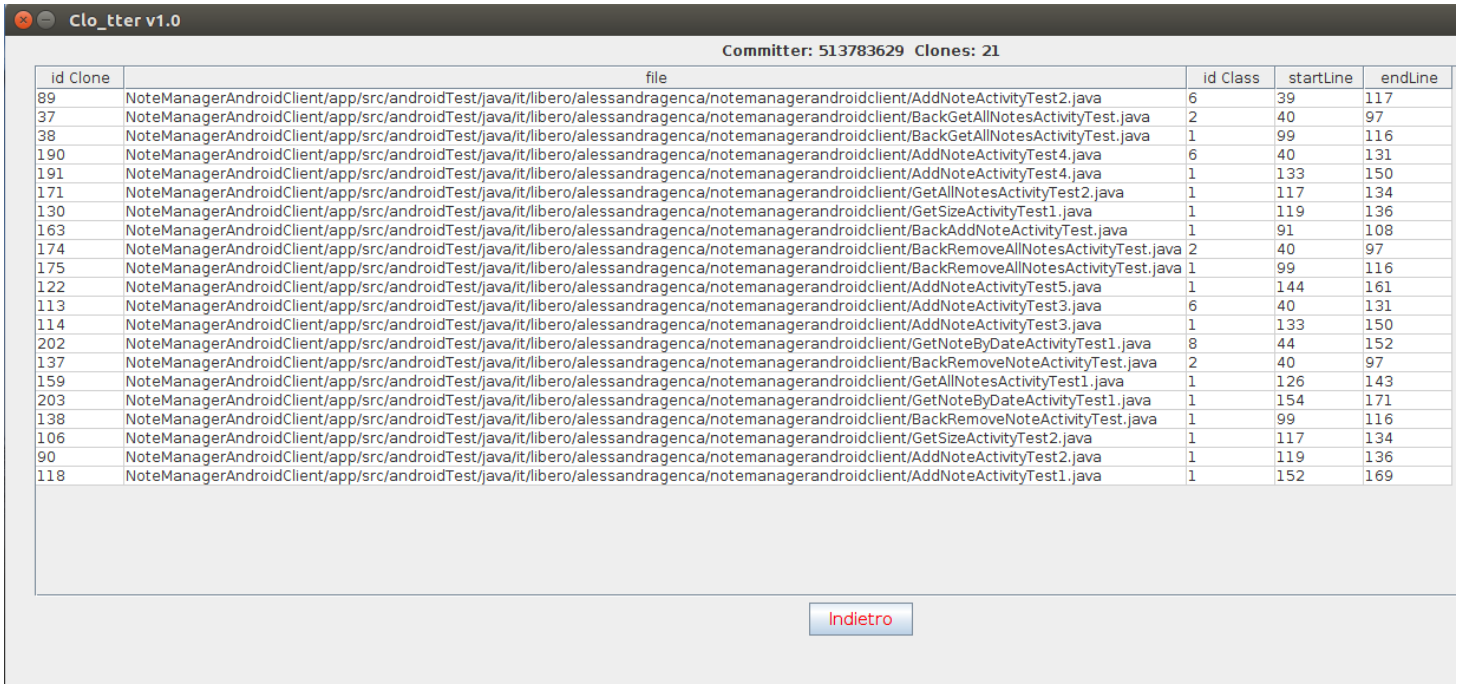


Figura 5.11 – Interfaccia di selezione di un clone

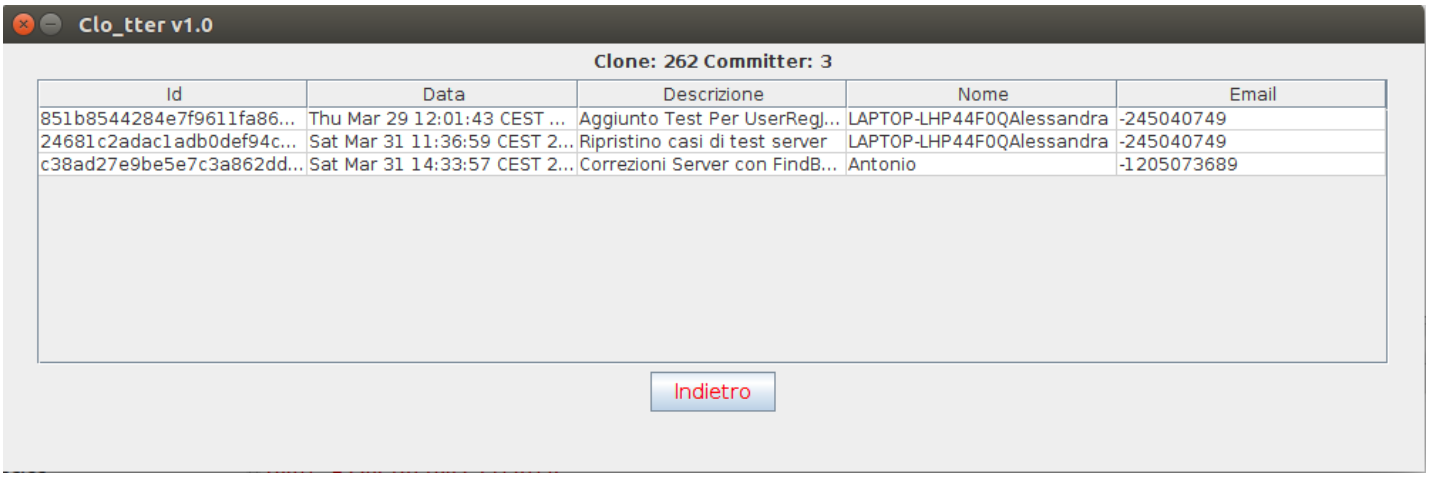
L’immagine 5.12 riporta tutti i cloni associati ad un committer precedentemente scelto tramite l’interfaccia della figura 5.10.



id Clone	file	id Class	startLine	endLine
89	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/AddNoteActivityTest2.java	6	39	117
37	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/BackGetAllNotesActivityTest.java	2	40	97
38	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/BackGetAllNotesActivityTest.java	1	99	116
190	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/AddNoteActivityTest4.java	6	40	131
191	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/AddNoteActivityTest4.java	1	133	150
171	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/GetAllNotesActivityTest2.java	1	117	134
130	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/GetSizeActivityTest1.java	1	119	136
163	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/BackAddNoteActivityTest.java	1	91	108
174	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/BackRemoveAllNotesActivityTest.java	2	40	97
175	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/BackRemoveAllNotesActivityTest.java	1	99	116
122	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/AddNoteActivityTest5.java	1	144	161
113	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/AddNoteActivityTest3.java	6	40	131
114	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/AddNoteActivityTest3.java	1	133	150
202	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/GetNoteByDateActivityTest1.java	8	44	152
137	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/BackRemoveNoteActivityTest.java	2	40	97
159	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/GetAllNotesActivityTest1.java	1	126	143
203	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/GetNoteByDateActivityTest1.java	1	154	171
138	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/BackRemoveNoteActivityTest.java	1	99	116
106	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/GetSizeActivityTest2.java	1	117	134
90	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/AddNoteActivityTest2.java	1	119	136
118	NoteManagerAndroidClient/app/src/androidTest/java/it/libero/alessandragenca/notemanagerandroidclient/AddNoteActivityTest1.java	1	152	169

Immagine 5.12 – Cloni associati a un committer

L’immagine 5.12 riporta tutti i committer associati al clone selezionato precedentemente tramite l’interfaccia di figura 5.11.



Id	Data	Descrizione	Nome	Email
851b8544284e7f9611fa86...	Thu Mar 29 12:01:43 CEST ...	Aggiunto Test Per UserRegI...	LAPTOP-LHP44F0QAlessandra	-245040749
24681c2adac1adb0def94c...	Sat Mar 31 11:36:59 CEST 2...	Ripristino casi di test server	LAPTOP-LHP44F0QAlessandra	-245040749
c38ad27e9be5e7c3a862dd...	Sat Mar 31 14:33:57 CEST 2...	Correzioni Server con FindB...	Antonio	-1205073689

Immagine 5.12 – Committers associati a un clone

5.5 Applicazione dell’approccio

L’approccio definito è stato applicato, in via sperimentale, a tre progetti presenti in GitHub (NoteManager, Dnsjava, e Tika), anche con lo scopo di verificare e validare la realizzazione del tool Clo_tter.
Nel seguito sono riportati, per brevità, i risultati ottenuti per due di tali sistemi:

- NoteManager: progetto formato da 16204 LOC, 321 file sorgenti e con dimensione pari a 17,8 MB;
- Dnsjava: progetto formato da 39490 LOC, 247 file sorgenti e con dimensione pari 27,1 MB.

Nel progetto NoteManager, l'analisi dei cloni effettuata da NiCad ha individuato 71 cloni suddivisi in 9 classi:

● Classe 1 = numero di cloni: 44; numero di linee: 13; similarità: 100%
● Classe 2 = numero di cloni: 7; numero di linee: 17; similarità: 82%
● Classe 3 = numero di cloni: 3; numero di linee: 20; similarità: 80%
● Classe 4 = numero di cloni: 5; numero di linee: 20; similarità: 80%
● Classe 5 = numero di cloni: 2; numero di linee: 19; similarità: 94%
● Classe 6 = numero di cloni: 3; numero di linee: 19; similarità: 85%
● Classe 7 = numero di cloni: 2; numero di linee: 19; similarità: 80%
● Classe 8 = numero di cloni: 2; numero di linee: 25; similarità: 84%
● Classe 9 = numero di cloni: 3; numero di linee: 13; similarità: 92%

Nella classe 1 sono stati rilevati cloni di tipo 1, cioè cloni esatti, nella classe 5 e 9 1 sono stati rilevati cloni di tipo 2, cioè cloni rinominati, e nelle classi 2, 3, 4, 6, 7 e 8 1 sono stati rilevati cloni di tipo 3, cioè near-miss clones.

Nel progetto Dnsjava, l'analisi dei cloni effettuata da NiCad ha individuato 44 cloni suddivisi in 20 classi:

- Classe 1 = numero di cloni: 2; numero di linee: 13; similarità: 85%
- Classe 2 = numero di cloni: 2; numero di linee: 13; similarità: 100%
- Classe 3 = numero di cloni: 2; numero di linee: 11; similarità: 100%
- Classe 4 = numero di cloni: 2; numero di linee: 16; similarità: 100%
- Classe 5 = numero di cloni: 2; numero di linee: 18; similarità: 83%
- Classe 6 = numero di cloni: 2; numero di linee: 15; similarità: 80%
- Classe 7 = numero di cloni: 2; numero di linee: 12; similarità: 91%
- Classe 8 = numero di cloni: 2; numero di linee: 19; similarità: 84%
- Classe 9 = numero di cloni: 2; numero di linee: 11; similarità: 90%
- Classe 10 = numero di cloni: 2; numero di linee: 11; similarità: 81%
- Classe 11 = numero di cloni: 2; numero di linee: 11; similarità: 81%
- Classe 12 = numero di cloni: 2; numero di linee: 11; similarità: 81%
- Classe 13 = numero di cloni: 2; numero di linee: 17; similarità: 82%
- Classe 14 = numero di cloni: 3; numero di linee: 15; similarità: 86%
- Classe 15 = numero di cloni: 2; numero di linee: 12; similarità: 83%
- Classe 16 = numero di cloni: 3; numero di linee: 16; similarità: 100%
- Classe 17 = numero di cloni: 2; numero di linee: 10; similarità: 80%
- Classe 18 = numero di cloni: 2; numero di linee: 20; similarità: 80%
- Classe 19 = numero di cloni: 2; numero di linee: 13; similarità: 92%
- Classe 20 = numero di cloni: 4; numero di linee: 10; similarità: 80%

Nella classi 2,3,4,16 1 sono stati rilevati cloni di tipo 1, nelle classi 7,9 e 19 sono stati rilevati cloni di tipo 2, nelle classi 1, 5, 6, 8 10, 11, 12, 13, 14, 15, 117, 18, 20 sono stati rilevati cloni di tipo 3.

Per il progetto NoteManager sono stati analizzati 133 commits effettuati da 3 committer.
Per il progetto Dnsjava sono stati analizzati 1657 commits effettuati da 2 committer.

L'analisi di Clo_tter per NoteManager ha prodotto 75 coppie cloni/committer, il numero di cloni massimo associato ad un committer è stato di 49 cloni associati al committer con id: 1205073689, quello più basso di 2 cloni associati al committer con id: 245040749.
Sempre per NoteManager, il numero più alto di committer associato ad un clone è stato di 4 committer associati al clone con id: 262, quello più basso di 1 committer per la maggior parte dei cloni.

L'analisi di Clo_tter per Dnsjava ha prodotto 44 coppie cloni/committer, tutti associati al committer bwelling con id: 1594128718.

Conclusioni

Lo scopo del presente lavoro di tesi è stato il recupero, l'analisi e l'elaborazione di dati registrati in repository di progetti software open-source per individuare in essi la presenza di cloni software ed associarli ai relativi committers.

A tal fine è stata definita una tool-chain che permette di fare; il download di un sistema software dal sistema di hosting di progetti GitHub, scaricandone i file del codice sorgente e dei commit effettuati; l'analisi dei sorgenti per identificare i cloni in ess; l'analisi dei commit e individuare le associazioni tra cloni e committer che su quei frammenti hanno operato e quindi probabili responsabili dell'introduzione di quei cloni.

In particolare, è stato sviluppato il tool, Clo_tter, per effettuare l'analisi e la sintesi delle informazioni relative ai commit e associa tali informazioni a quelle ottenute dal tool NiCad, relative ai cloni individuati.

La principale complessità di tale sviluppo è stata dovuta al recupero delle informazioni associate ai progetti relative ai commits e ai cloni e dalla differente tipologia e formato della rappresentazione dei dati recuperati.

Sono stati effettuati alcuni esperimenti utilizzando sistemi software open source prelevati di GitHub per verificare e validare il tool Clo_tter, che è risultato rispondere in pieno ai requisiti per esso specificati.

Relativamente alle informazioni prodotte dal tool relative all'associazione tra i cloni identificati ed i committer, esse sono un primo utile risultato per le successive analisi da effettuare per individuare i possibili committer che hanno introdotto quei cloni.

L'attuale implementazione presenta un limite principale dovuto al fatto che è analizzata una sola release del sistema considerato e ciò non è sufficiente a conoscere quando un clone è stato introdotto nel sistema, e quindi non necessariamente il committer ad esso associato è colui che lo ha introdotto.

Uno sviluppo futuro quindi sarà analizzare più release dello stesso progetto software in modo tale da riuscire ad associare in modo più preciso il clone al committer che

effettivamente lo aveva introdotto. Questa analisi più completa delle varie versioni permetterà anche di analizzare l'evoluzione dei cloni nel sistema rispetto ai vari committer.

Bibliografia

[Baxter] Baxter, I.D., Yahin, A., Moura, L., Sant'Anna, M., Bier, L.: Clone Detection Using Abstract Syntax Trees. In Koshgoftaar, T.M., Bennett, K., eds.: International Conference on Software Maintenance, IEEE Computer Society Press (1998) 368–378

[Kaur] Geetika Chatley, Sandeep Kaur and Bhavneesh Sohal "Software Clone Detection: A review", International Journal of Control Theory and Applications · January 2016, International Science Press

[Koske] Rainer Koschke. Survey of Research on Software Clones. In Proceedings of Dagstuhl Seminar 06301: Duplication, Redundancy, and Similarity in Software, 24pp., Dagstuhl, Germany, July 2006

[Nicad] Cordy, James R., and Chanchal K. Roy. "The NiCad clone detector." *Program Comprehension (ICPC), 2011 IEEE 19th International Conference on*. IEEE, 2011.

[Git] https://www.git-scm.com/

[GitHub] https://github.com/

[Sourcetree] https://www.sourcetreeapp.com/