Università degli studi di Salerno

Progetto di Ingegneria del Software II

Repominer Evolution

Documento di manutenzione: analisi

Autori:
Matteo Merola
Carlo Branca
Simone Scalabrino
Giovanni Grano

Supervisore:
Prof. Andrea De Lucia

Documento di Manutenzione: analisi
Versione 2.0

25 maggio 2014

Revision History

Tabella 1: Tabella delle revisioni del documento

Data	Versione	Descrizione	Autori
19/05/2014	1.0	Versione iniziale	Simone Scalabrino, Giovanni Grano, Carlo Branca, Matteo Merola
25/05/2014	2.0	Modificata Impact Analysis	Simone Scalabrino, Giovanni Grano, Carlo Branca, Matteo Merola

Indice

1 Introduzione **4**
1.1 Scopo del sistema 4
1.2 Definizioni, acronimi, abbreviazioni 4
1.3 Panoramica 5
1.4 Sistema corrente 7

2 Sistema proposto **10**
2.1 Requisiti funzionali 10

3 System Model **12**
3.1 Use Case Models 12
3.2 Object Models 15

4 Impact Analysis **16**

Elenco delle figure

3.1 UC QualityMetricsExtractor 12
3.2 UC ChangeConfigSettings 14
3.3 Modello a oggetti 15

4.1 Dettaglio delle relazioni da instaurare 17
4.2 Schema relazionale del DB con le relazioni princpalmente interessate dall'analisi 18

1. Introduzione

Il presente documento si propone di fornire una rapida panoramica del progetto, focalizzandosi sui requisiti funzionali da implementare a partire da una base già esistente. Si provvederà a specificare un design di alto livello delle funzionalità che si andranno ad aggiungere. In questo documento andrà a confluire anche tutta l'attività di impact analysis.

1.1. Scopo del sistema

Scopo del sistema è quello di aggiungere nuove features ad un sistema già esistente, RepoMiner. Tale sistema è in grado di analizzare codice sorgente ed estrarre da esso una serie di metriche.

Si vuol essere in grado con il sistema proposto, di sfruttare come base di conoscenza le informazioni estratte da RepoMiner per calcorare il grado di entropia presente nei package di un sistema software. In questo senso, l'entropia può essere calcolata implementando una serie di metriche, calcolate dagli sviluppatori nell'ambito dell'evoluzione del sistema. Per una descrizione precisa di tale metriche si farà riferimento alla sezione 2.1 del presente documento. Attraverso questi parametri sarà possibile rappresentare l'informazione di un particolare package.

Il secondo step del progetto consiste nell'implementazione del *Basic Code Change Model* e nell'eventuale implementazione dell'*Extended Code Change Model*, descritti nell'articolo di Hassan[1]. Per una descrizione più accurata di fa ancora una volta riferimento alla sezione 2.1 di tale documento.

1.2. Definizioni, acronimi, abbreviazioni

Entropia: in meccanica statistica, l'entropia rappresenta una grandezza che viene interpretata come una misura del disordine presente in un sistema fisico, incluso, nel caso limite, l'universo. L'entropia è una funzione crescente

della probabilità dello stato macroscopico di un sistema, e precisamente risulta proporzionale al logaritmo del numero delle configurazioni microscopiche possibili per quello stato macroscopico: la tendenza all'aumento dell'entropia di un sistema isolato corrisponde dunque al fatto che il sistema evolve verso gli stati macroscopici più probabili.

FEATURE: una feature è un insieme di requisiti logicamente correlati. In un prodotto software, è un servizio o una funzionalità offerta.

BUG: un bug è una fault in un programma, che fa si che il programma metta in atto un comportamento inatteso o che esegua delle operazioni non intenzionali.

1.3. Panoramica

In riferimento al ciclo di vita del software, è chiaro osservare come, una volta che il prodotto viene rilasciato, andrà incontro per forza di cose a diverse modifiche per i più svariati motivi, dalla correzione di errori all'estensione di funzionalità. Tale concetto viene espresso dalla prima legge di Lehman dulla manutenzione del software:

un sistema deve necessariamente cambiare ed evolvere con continuità per mantenere intatto il suo livello di utilità, stante i continui cambiamenti che avvengono nell'ambiente in cui il sistema opera.

Quando un sistema cambia, per qualsivoglia motivo, la tua struttura tende a diventare più complessa. Diventa necessario utilizzare risorse supplementari per preservarne la qualità della struttura. Assume importanza estrema quindi il monitorare la qualità del codice attraverso l'utilizzo di metriche.

Nell'Ingegneria del software, la predizione dei faults è stata sempre comunemente associata a misurazioni di complessità. Il concetto di entropia più essere applicato con successo in diversi contesti, a partire dal concetto di defect prediction. L'idea di base è quella di proporre metriche complesse che sono basate sul processo di cambiamento del codice invece che sul codice stesso. Si può ragionevolemente congetturare come, un processo di cambiamento complesso e disomogeneo, possa affliggere negativamente un sistema software.

Proprio come strumento per quantificare la complessità ritorna utile il concetto di entropia di Shannon. A partire da queste informazioni è possibile andare ad implementare due modelli per la complessità dei cambiamenti sul

codice; il *Basic Code Change Model* e un modello più elaborato e complesso, il *Extended Code Change Model*. Entrambi questi modelli vanno a calcolare un singolo valore che misura la complessità complessiva dei cambiamenti di un progetto durante un periodo di tempo stabilito.

Come mostrato nel caso di studio svolto nel lavoro di Hassan [1], tali metriche basate sulla complessità dei cambiamenti si rivelavano estremamente valide nella defeat prediction.

Occore specificare come, in questi modelli, si andranno ad analizzare solamente le cosiddette FI, ossia *Features Introduction Modifications*; per FI intendiamo modifiche al codice che vanno ad implementare nuove features nel sistema. In questa categoria andrà a confluire tutto ciò che non può essere catalogato come FR (*Fault Repairing Modifications*) e come GM (*General Maintenance Modifications*). Come unità da misurare si è scelto il singolo file, in quanto si può ragionevolmente credere che è nel file che i developers concentrano e raggruppano le singole entità.

Come già detto, il modello BCC utilizza un arco temporale ben specificato, e definisce, file per file, la probabilità che lo stesso subisca un cambiamento. Più le probabilità sono equamente distribuite, più alta è l'entropia del sistema software, e dunque maggiore è la probabilità di svilupare difetti. Come caso limite invece, qualora un singolo file raggiunga probabilità 1, abbiamo che l'entropia del sistema diviene minima. L'idea chiave dunque risiede nel focalizzare l'analisi sulle modifiche alla singole linee di codice lungo un intervallo di tempo, allo scopo di costruire un modello di probabilità dei cambiamenti.

Quest'analisi fornisce ovviamente anche un andamento evolutivo di quella che è l'entropia del sistema lungo tutto il suo ciclo di vita. Partizionando lo stesso in periodi successivi di tempo, è interessante notare come il processo di cambiamento del codice si sia evoluto e protratto nel tempo di pari passo all'entropia dello stesso. Ciò costituisce anche un'importante parametro di monitoraggio per un manager. Si è osservato come, individuando le ragioni di picchi inaspettati nell'entropia, sia possibile pianificare ed essere pronti per problemi futuri.

Nel BCC sin qui descritto a grandi linee si assume un intervallo di tempo fissato per il calcolo dell'entropia, e si assume che il numero di file in un sistema rimanga inalterato. Ovviamente queste limitazioni restringono in campo di utilizzo di questo modello. L'*Extended Code Change* va a colmare tali lacune. In particolare, per quanto riguarda l'arco di tempo da analizzare nell'evoluzione software, è possibile prendere in considerazione diverse

opzioni differenti. Abbiamo già visto il criterio del *time based periods*, ossia la suddivisione in parti eque dei periodi di tempo, a partire dall'inizio del progetto fino alla sua conclusione. Si crede che un periodo di periodo di un quarto di anno sia un limite di tempo ragionevole per raccogliere una frazione significativa nella crescita di un sistema. Un'altra possibilità riguarda i *modification limit base periods*, ossia il suddividere il tempo in periodi basandosi sul numero di modifiche che sono memorizzare nella repository in oggetto. Per prevenire i casi in cui si verificano lunghi periodi di scarse modifiche implementative, è ragionevole imporre un limite temporale di 3 mesi qualora un periodo non raggiunga un limite fissato a 600 modifications. In conclusione, si è osservato come il processo di modifica segua delle fasi di picco, seguire da fasi di rilassamento. È sulla base di questa osservazione che è possibile la suddivisione temporale in *burst based periods*.

Nel modello ECC è necessario definire un'entropia normalizzata, con lo scopo di comparare l'entropia di sistemi di differente dimensione, con aggiunta o rimozione di files durante i vari periodi di tempo. La *normalized static entropy H* dipende dal numero di files in un sistema. È interessante notare alcuni sistemi software annoverino alcuni files con una probabilità di cambiamento davvero bassa. Per fare in modo che questi files non riducano in maniera artificiosa l'entropia del sistema, si è definita l'*adaptive sizing entropy*. L'idea di fondo è quella di dividere per il numero di linee recentemente modificate, invece per il numero attuale di linee del sistema. Vi sono due criteri di scelta per calcolare tale valore, ossia un primo criterio che prende in esame i files modificati nei precedenti x mesi,incluso il mese corrente, mentre un secondo prevede che il set di files da analizzare includa tutti i files modificati nei precedenti x periodi, incluso quello corrente.

1.4. Sistema corrente

RepoMiner è un lavoro di tesi triennale sviluppato da alcuni studenti dell'Univerisità degli Studi di Salerno. RepoMiner offre alcuni strumenti per la raccolta automatica di informazioni da un specifica repository. L'architettura del sistema si compone di tre moduli fondamentali.

QualityMetricsExtractor: si occupa di analizzare il codice sorgente per estrarre una serie di specifiche metriche software. Abbiamo la possibilità di estrarre quindi informazioni su classi, dimensione, dipendenze. Tale modulo è composto da un parser java e da un estrattore di metriche che utilizza i dati forniti dal parser per calcolare alcune metriche di qualità del software:

- Lines Od Code (LOC): calcola il numero di linee di codice per una data classe
- Cyclomatic Complexity (CC): calcola la complessità di una data classe considerando le sue strutture di controllo
- Lack of Cohesion of Methods (LCOM): calcola la coesione della classe considerando il numero di variabili che i metodi condividono
- Conceptual Cohesion of Class (C3): calcola la coesione concettuale di una classe
- Response for a Class (RFC): esprime il numero di servizi che la classe è in grado di fornire alle altre classi
- Number of Children (NOC): calcola il numero di classi figlie immediatamente sotto la classe padre nella gerarchia
- Coupling Between Object Classes (CBO): calcola l'accoppiamento tra le classi di un sistema
- Depth of Inheritance Tree (DIT): calcola la massima distanza di una classe dalla radice dell'albero

GitExtractor: si occupa dell'estrazione di informazioni dal sistema di versioning GIT, dal quale è possibile estrarre anche i singoli cambiamenti tramite i file di log estraibili. Il tool si compone di un estrattore che scarica codice sorgente e log dei cambiamenti dal repository, e da un parser che usa il log per estrarre i contenuti di ogni singolo commit e le informazioni che comprendono classi e metodi modificati tra un commit e l'altro. In particolare, le informazioni specifiche sono le seguenti:

- Identificativo del cambiamento
- Data della modifica
- Identificativo dello sviluppatore che ha apportato la modifica
- Email dello sviluppatore
- Commenti lasciati dallo sviluppatore
- Lista di File modificati
- Metodi modificati per ogni classe

BugZillaExtractor: analizza i sistemi di Issue Tracker. Da qui è possibile estrarre informazioni sui bug-fixes, la data di risoluzione, gli sviluppatori coinvolti nella discussione e nella risoluzione del bug e file coinvolti. BugZillaExtractor è composto da due moduli, uno che estrrare la lista dei bug segnalati, mentre il secondo analizza questa lista per estrarre informazioni su ogni singolo bug.

2. Sistema proposto

2.1. Requisiti funzionali

Il sistema dovrà essere in grado di calcolare metriche relative all'entropia di un progetto software.

Più in particolare, dovrà essere possibile calcolare le seguenti metriche basilari:

- Numero di revisioni del sistema
- Numero medio di volte in cui i file di un package hanno subito cambiamenti
- Numero medio di volte in cui i file di un package hanno subito operazioni di refactoring
- Numero medio di volte in cui i file di un package hanno subito operazioni di bug fixing
- Numero di autori di commit effettuati all'interno di un package
- Numero di linee aggiunte o rimosse (somma, media e massimo)
- Dimensione media dei file modificati

Il sistema dovrà implementare il Basic Code Change Model descritto nell'articolo di Hassan [1]. In primo luogo, dovranno essere estratti i cambiamenti dovuti ad aggiunte di feature: per fare ciò si analizzeranno i messaggi di commit (non dovranno contenere parole che denotano altri tipi di cambiamenti, come, ad esempio: bug-fix, re-indent, copyright update, etc.).

Dovranno essere implementate, quindi, le seguenti metriche:

- Numero di cambiamenti totali di ogni file (righe aggiunte + righe cancellate) di un dato progetto software a seguito di aggiunta di feature in un periodo di riferimento (es: un mese)

- Entropia dei cambiamenti dei file del progetto nel periodo fissato

Il sistema dovrà implementare l'Extended Code Change Model descritto nell'articolo di Hassan [1]; l'utente dovrà poter scegliere la tipologia di suddivisione del sistema nel tempo tra:

- **Periodi fissati**: i cambiamenti del sistema sono divisi in periodi di tempo prestabiliti (es:un mese), come nel BCC
- **Numero di cambiamenti fissati**: si usa un numero prefissato di cambiamenti per stabilire i differenti periodi da analizzare
- **Periodi di burst**: si distinguono i periodi in base alle raffiche di cambiamenti che occorrono

Inoltre, il sistema calcolerà, al posto dell'entropia classica, l'Adapting Sizing Entropy, ovvero l'entropia normalizzata in base al numero di file modificati nell'ultimo periodo (che dovrà essere definito dall'utente, e potrà essere *temporale* o basato sul *numero di modifiche*).

3. System Model

3.1. Use Case Models

3.1.1. UC_QualityMetricsExtractor

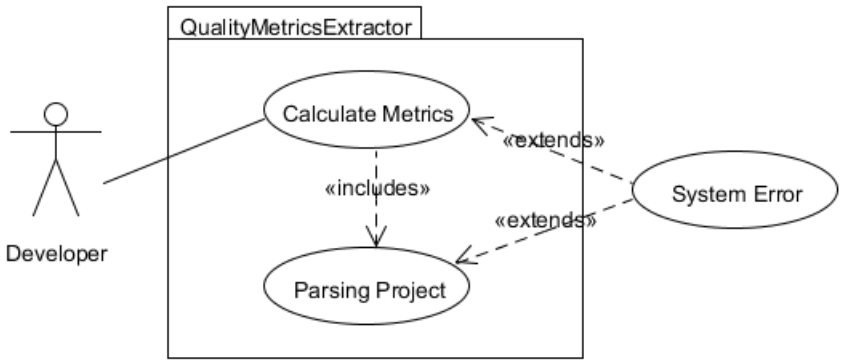

Figura 3.1: UC QualityMetricsExtractor

Nome Use Case	**Calcolo metriche**
ID:	UC_QualityMetricsExtractor
Partecipanti:	Sviluppatore
Condizioni di ingresso:	Lo sviluppatore avvia la IDE "Eclipse".

Flusso di eventi:

1. Lo sviluppatore avvia il tool tramite il tasto "run".
2. "SIE" apre una finestra della IDE mostrando sulla sinistra la lista dei progetti che è possibile analizzare.
3. Lo sviluppatore seleziona il progetto, e da' avvio al programma.
4. "SIE" notifica l'avvenuto calcolo delle metriche tramite un messaggio che compare in console.

Condizioni di uscita:

a. Il sistema memorizza i dati.

b. Lo sviluppatore termina precocemente l'operazione di calcolo.

Eccezioni:	Errore di sistema

3.1.2. UC_ChangeConfigSettings

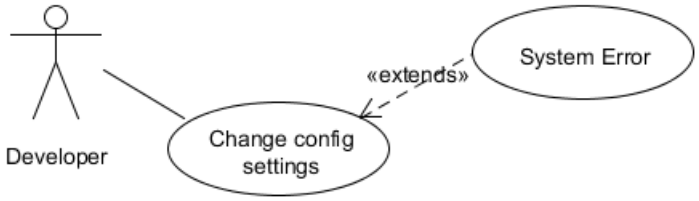

Figura 3.2: UC ChangeConfigSettings

Nome Use Case	**Modifica file di configurazione**
ID:	UC_ChangeConfigSettings
Partecipanti:	Sviluppatore
Condizioni di ingresso:	Il plugin è installato correttamente nell'IDE "Eclipse".
Flusso di eventi: 1. Lo sviluppatore apre il file settings.config con un qualsiasi editor di testo. 2. Lo sviluppatore configura le metriche da calcolare.	
Condizioni di uscita: a. Il plugin è configurato con le metriche desiderate. b. Lo sviluppatore ha scritto male i dati di configurazione.	
Eccezioni:	Errore di sistema

3.2. Object Models

La struttura dell'intero sistema software si sintetizza nel modello a oggetti rappresentato dal seguente diagramma.

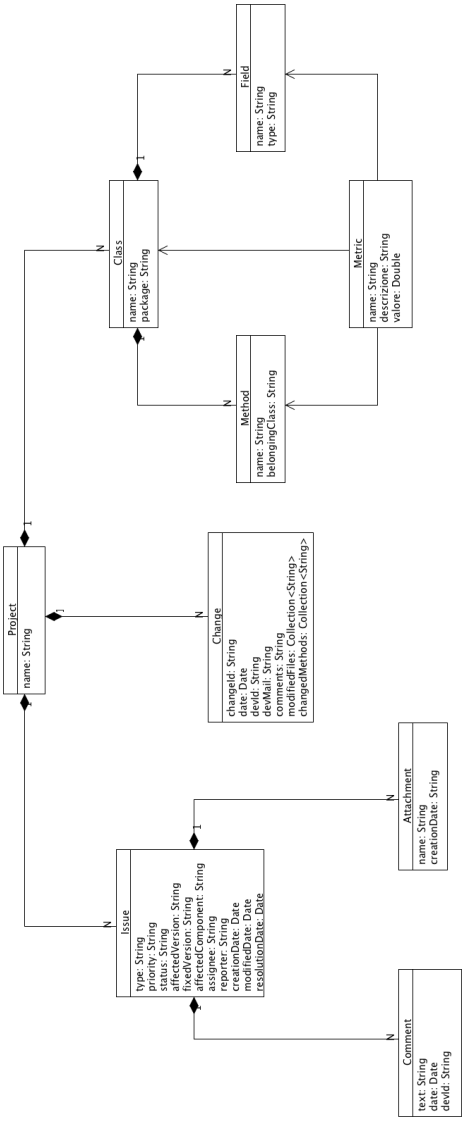

Figura 3.3: Modello a oggetti

4. Impact Analysis

In questa sezione si presenterà una previsione dell'impatto che la modifica richiesta avrà sugli artefatti del sistema esistente. In particolar modo si cercherà di individuare le classi (codice sorgente) che dovranno essere modificate per integrare le funzionalità richieste.

Dato che si è scelto di implementare un sistema ex novo le cui funzionalità andranno ad affiancare e integrare quelle del sistema esistente, a livello di codice sorgente l'impatto sarà pari a zero.

RepominerEvo andrà ad interagire con esso solamente al layer di gestione dei dati, in quanto andrà a utilizzare le informazioni estrapolate e memorizzate da Repominer per implementare nuove funzionalità. Si può ragionevolmente dedurre quindi che l'impatto sul sistema esistente andrà gestito unicamente a livello di database.

Dato che dovranno essere implementate nuove metriche di natura più complessa ed articolata, si presume che sarà necessaria la modifica di alcune tabelle e di alcuni campi nel database esistente.

Nella figura 4.2 di pagina 18 riportiamo la relazioni e le tabelle più significative al momento corrente ai fini delle metriche da implementare.
In particolare, ipotizziamo che lo Starting Impact Set conterrà la tabella:

- *'metrics'*

Si è pensato, dovendo estendere le metriche già calcolate in Repominer con nuove metriche, distinte in metriche di pacchetto e metriche di progetto, di dover prevedere la creazione di due nuove tabelle:

- *'package_metrics'*
- *'project_metrics'*

Nella figura 4.1 di pagina 17 mostriamo le nuove relazioni ipotizzate che si verranno a creare. Si precisa che tali nuove relazioni non avranno alcuna influenza con la struttura del database preesistente.

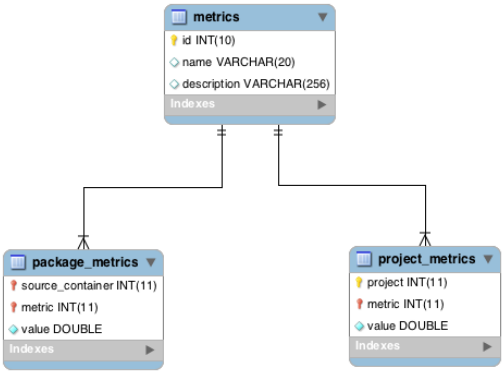

Figura 4.1: Dettaglio delle relazioni da instaurare

A seguito di un'analisi approfondita, è stato individuato il Candidate Impact Set nell'unica tabella:

- *'metrics'*

A seguito di tale analisi rimangono valide le considerazioni sopra esposte riguardo le nuove tabelle da creare e le relazioni da instaurare con *'metrics'*.

Una valutazione dettagliata dei risultati di questa analisi sarà presentata in un report separato dopo l'implementazione completa delle modifiche.

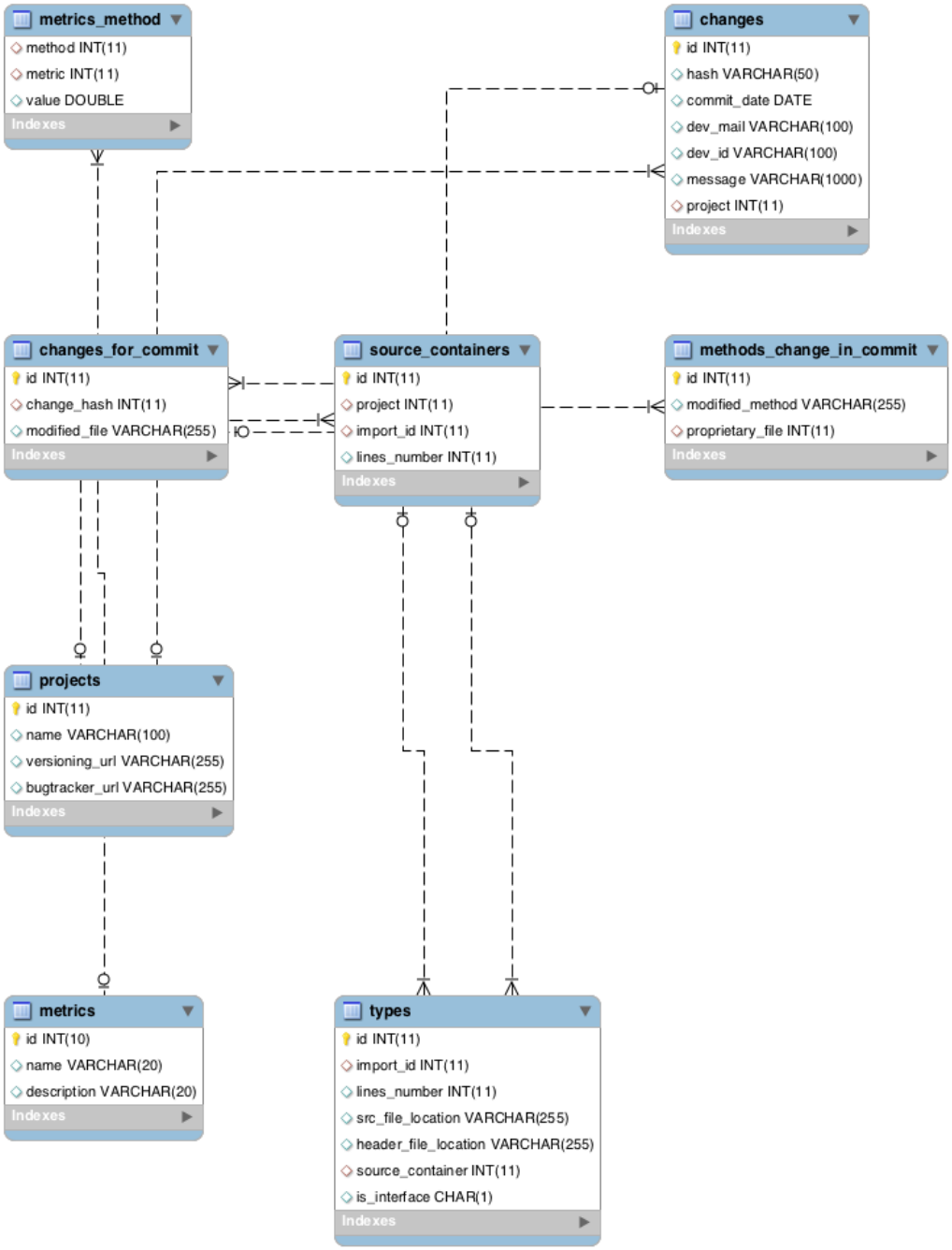

Figura 4.2: Schema relazionale del DB con le relazioni princpalmente interessate dall'analisi

Bibliografia

[1] Ahmed E Hassan. Predicting faults using the complexity of code changes. In *Proceedings of the 31st International Conference on Software Engineering*, pages 78–88. IEEE Computer Society, 2009.