IDEAL Reverse Engineering Document

Corso di Ingegneria Gestione ed Evoluzione del Software - IDEAL

Domenico Antonio Gioia
d.gioia7@studenti.unisa.it
Matricola: 0522501518

Giovanni Scorziello
g.scorziello5@studenti.unisa.it
Matricola: 0522501701

Antonio Scognamiglio
a.scognamiglio32@studenti.unisa.it
Matricola: 0522501496

Link Repository GitHub

1 INTRODUZIONE

In questo documento vengono presentati gli artefatti di reverse engineering estratti a partire da un'analisi del sistema IDEAL. Per una maggiore efficienza sono stati realizzati solo gli artefatti ritenuti più significativi nella rappresentazione del sistema a vari livelli di astrazione e relativi alle funzionalità più complesse. Gli artefatti inclusi a partire dal Capitolo 3 comprendono:

- 3 *use case*
- 3 *sequence diagram*
- 1 *package diagram*
- 1 *class diagram*

Il progetto, inoltre, contiene già della documentazione visionabile a questo link, in cui sono listati i linguistic antipattern che analizza, l'architettura del sistema, alcune istruzioni per l'installazione e l'utilizzo, e dei dettagli sui vincoli a cui è sottoposto.

2 PANORAMICA DEL SISTEMA

Il sistema è progettato per assistere gli sviluppatori nello sviluppo di programmi Java e C#. Può analizzare i file di codice sorgente per individuare i linguistic antipattern definiti da [1]. L'utente può analizzare cartelle intere o specifici file. Vengono controllati elementi come classi, metodi e variabili e le violazioni alle regole vengono registrate in un file CSV.
Le change request proposte, includono di permettere al software di analizzare file di codice sorgente Python, oltre che semplificare le fasi di installazione, configurazione, ed esecuzione del tool.

3 MODELLI DEL SISTEMA

3.1 Use Case Models

(1) IDEAL-UC-1
(2) IDEAL-UC-3
(3) IDEAL-UC-4

3.2 Sequence Diagrams

In questa sezione vengono presentati in dettaglio i sequence diagram generati durante il processo di reverse engeneering. Tutti i sequence diagram prodotti sono mappati ad uno specifico use case e rappresentano i flussi di esecuzione normale (senza errori).

3.2.1 Identificazione di linguistic antipattern in file di codice Java. In questo sequence diagram viene mostrato come vengono utilizzate le varie classi per procedere all'identificazione di linguistic antipattern in codice sorgente Java. Dopo aver istanziato il model per la configurazione vengono creati vari controller per eseguire il *part-of-speech tagging*, e lo *splitting* del linguaggio naturale. Il controller per il salvataggio dei risultati, *ResultsWriter*, poi, tramite il controller per l'analisi invoca il metodo *analyze* per eseguire le varie operazioni di analisi delle regole per i linguistic antipattern; il controller *Main*, infine, invoca il metodo *save_issues* di *ResultsWriter* per salvare i risultati in modo persistente.
Link al sequence diagram: IDEAL-SD-1

3.2.2 Lettura file da analizzare. In questo sequence diagram viene mostrato il processo tramite il quale viene preso in input il progetto per il quale si vuole verificare la presenza linguistic antipattern. Viene istanziato un oggetto di tipo project, tramite il quale viene letto il file di configurazione. All'interno di esso, vi sono tutte le regole necessarie al sistema per effettuare un corretto parsing del progetto, incluse regole custom inserite dall'utente, come per esempio eventuali termini da ignorare, sinonimi utilizzati ecc...
Successivamente vengono presi in input i file del progetto da analizzare, tramite la path fornita dall'utente. Può essere considerato sia un singolo file, sia un progetto vero e proprio, quindi composto da cartelle, sotto cartelle, ecc...
I File presi in input vengono memorizzati all'interno di un array, pronti per la fase di analisi. Essi dovranno avere un'estensione valida, ovvero deve trattarsi di file Java o C++.
Link al sequence diagram: IDEAL-SD-3

3.2.3 Lettura file di configurazione. Nel sequence diagram illustrato è delineato il processo attraverso il quale vengono recuperate le informazioni di configurazione necessarie per condurre un'analisi dei file allo scopo di individuare antipattern linguistici. Una volta avviata l'analisi, lo strumento procede al reperimento delle informazioni di configurazione denominate *output_directory*, *input_file*, *junit_version*, *custom_code* e *custom_terms* all'interno del file di configurazione. Successivamente, acquisisce i percorsi relativi al modello del tagger, al file JAR del tagger Standford e al percorso dell'eseguibile Java. Tale fase avvia un processo che esegue operazioni di part-of-speech tagging per l'intera durata dell'analisi. Durante il momento di parsing di ogni singolo file, il sistema estrae il percorso dell'eseguibile *srcML* dal file di configurazione. Una volta completata l'analisi, il processo di *part-of-speech tagging* viene concluso e terminato.
Link al sequence diagram: IDEAL-SD-4

3.3 Package Diagram

Il progetto è costituito da sei pacchetti principali:

- **apps**, il quale è responsabile dell'esecuzione dell'analisi dello strumento.
- **model**, il quale gestisce il recupero delle informazioni di configurazione e dei risultati dell'analisi.
- **common**, responsabile di varie funzioni di utilità.
- **service**, incaricato delle operazioni di parsing dei file e della lettura delle configurazioni.

- **rule**, responsabile dell'individuazione di specifici antipattern linguistici nel codice.
- **nlp**, responsabile del Part-of-Speech Tagging e della Sentence Splitting.

Link al package diagram: IDEAL-PACKAGEDIAGRAM

3.4 Class Diagram

Le classi estratte e contenute nel class diagram sono state utilizzate nei sequence diagram illustrati nel Paragrafo 3.2.
Link al class diagram: IDEAL-CLASSDIAGRAM

REFERENCES

[1] Venera Arnaoudova, Massimiliano Di Penta, and Giuliano Antoniol. "Linguistic antipatterns: What they are and how developers perceive them". In: *Empirical Software Engineering* 21 (2016), pp. 104–158.