**Il tool SURFACE**

SURFACE è un tool scritto in Java sviluppato dal dott. Emanuele Iannone. Tale tool consente, mediante un’interazione da linea di comando, di calcolare metriche di sicurezza per codice Object-Oriented fornendo una panoramica del profilo di sicurezza delle classi presenti in un progetto. Può calcolare sia metriche relative a singole classi che ad un intero progetto, che a sua volta può essere sia locale (specificato da un path locale) che remoto (specificato dal link della repository del progetto e dall’hash dell’ultimo commit). I risultati del calcolo delle metriche di sicurezza possono essere mostrati su standard output ed esportati in CSV. Di seguito vengono elencate le metriche attualmente implementate all’interno del progetto, accompagnate da una breve descrizione:

**Metriche *Class-level***

* **CA (Classified Attributes)** - numero di variabili d’istanza (o attribuiti) che vengono identificati come “riservate” (dall’inglese, *classified*) mediante delle euristiche. Nello specifico, viene verificato il match dei nomi delle variabili con un insieme di parole che suggeriscono che quella variabile contenga un valore riservato;
* **CM (Classified Methods)** - numero di metodi che vengono identificati come “riservati”. Un metodo viene identificato come riservato se interagisce con almeno un *Classified Attribute;*
* **CIVA (Classified Instance Variables Accessibility)** - rapporto tra il numero di *Classified Attributes* che non sono né privati né staticied il numero totale di *Classified Attributes;*
* **CCVA (Classified Class Variables Accessibility)** - rapporto tra il numero di *Classified Attributes* che non sono privati ma sono staticied il numero totale di *Classified Attributes;*
* **CMA (Classified Method Accessibility)** - rapporto tra il numero di *Classified Methods* che non sono privati sul totale dei *Classified Methods;*
* **RP (Reflection Package)** - valore booleano che indica se la classe in esame fa uso del pacchetto *java.reflection;*
* **CMR (Classified Methods Ratio)** - rapporto tra il numero di *Classified Methods* e numero di metodi totali;
* **CAI (Classified Attribute Interactions)** - rapporto tra la somma del numero di *Classified Attributes* di ogni *Classified Methods* che li usa ed il prodotto tra tutti gli usi di *Classified Methods* e *Classified Attributes.*

**Metriche *Project-level***

* **CC (Critical Classes)** - numero di classi critiche all’interno del progetto. Una classe si definisce critica se e solo se ha almeno un *Classified Attribute* oppure un *Classified Method*;
* **CCR (Critical Classes Ratio)** - rapporto tra il numero di *Critical Classes* ed il numero totale di classi all’interno del progetto;
* **SCCR (Serializable Critical Classes Ratio)** - rapporto tra il numero di *Critical Classes* che implementano l’interfaccia *Serializable* ed il numero di *Critical Classes*;
* **CCE (Critical Classes Extensibility)** - rapporto tra il numero di *Critical Classes* non final (da cui quindi altre classi possono ereditare) ed il numero totale di *Critical Classes*;
* **CME (Classified Methods Extensibility)** - rapporto tra il numero di *Classified Methods* di cui è possibile effettuare l’override ed il numero di *Classfied Methods*;
* **CSCR (Critical Superclasses Ratios)** - media dei rapporti tra numero di *Critical Superclasses* e tutte le loro superclassi.

**Change Requests**

È stato deciso di apportare diverse modifiche al sistema attualmente disponibile. Nello specifico sono state definite 3 change requests:

1. Disaccoppiamento della gestione dell’input dall’interfaccia a caratteri. Attualmente il tool prende parametri solo da linea di comando ed il sistema di gestione dell’input non prevede la possibilità di usare altri metodi. Infatti, vi è una classe CliStarter che istanzia staticamente un CliParser che si occupa del parse dei parametri. Si prevede di dare la possibilità di rendere modulare la gestione dell’input. La priorità di tale intervento di modifica è alta.
2. Rendere la logica applicativa invocabile mediante servizi REST. Al momento l’interazione con l’applicazione avviene solo localmente installando il tool manualmente ed avviando il JAR. Si prevede di intervenire sulla logica di invocazione del tool per superare questo limite e renderlo invocabile anche in maniera remota. È stato deciso di limitare ai soli progetti remoti l’interazione con il tool mediante servizio REST.
3. Implementare un’interfaccia grafica che renda possibile l’utilizzo del tool mediante un Browser Web evitando di dover effettuare manualmente le chiamate agli endpoint REST.

**CR1: Idee per l’implementazione**Viene creata una classe astratta chiamata Parser che possiede un singolo metodo astratto parse e lo stato che prima apparteneva esclusivamente a CliParser. CliParser estenderà la classe astratta Parser. CliStarter non istanzierà un CliParser ma un Parser. Verrà usato il design pattern Abstract Factory e verrà introdotta una classe ParserFactory che conterrà un singolo metodo statico getParser che prende in input l’identificativo del parser che si vuole utilizzare e si occuperà di istanziarlo e restituirlo al chiamante. Questo parametro verrà passato alla classe main tramite argomenti CLI.

**CR2: Idee per l’implementazione**Viene creato un controller Spring che funge da endpoint REST. Da questo controller verrà richiamata la logica applicativa e l’input verrà fornito come JSON e restituirà, allo stesso modo, il risultato come JSON all’utente. Il JSON restituito viene prelevato da un file (in formato JSON) che verrà generato dal JSONExporter (classe che verrà creata durante l’implementazione di questa change request). Il controller dovrà fornire all’input di Surface “json” come exportFormat in maniera hard-coded. Non viene effettuata l’esportazione del file csv nel caso di progetti locali, ma ci si limita unicamente alla stampa.

**CR3: Idee per l’implementazione**Dal momento che con la CR2 verrà implementato un insieme di endpoint REST, è possibile sfruttare tali endpoint per creare una WebApp indipendente dal tool che consenta un’interazione semplificata con quest’ultimo disaccoppiando le tecnologie. Verrà, infatti, utilizzato il Framework React che mediante semplici chiamate HTTP ottiene i risultati dell’analisi del tool e le mostra all’utente.

**Impact Analysis**

In seguito alla definizione delle Change Requests è stata effettuata un’Impact Analysis iniziale volta all’identificazione dello **Starting Impact Set** e del **Candidate Impact Set**. Data l’assenza di documentazione, l’unico strumento disponibile per effettuare tale analisi è stato il codice, sul quale è stata effettuata in primo luogo una fase di code comprehension (il cui resoconto è consultabile nella sezione *docs* della Repository) e dal quale, successivamente, è stato ottenuto un Class Diagram utilizzando il tool di reverse engineering integrato nell’IDE IntelliJ IDEA. In una fase successiva, a partire dal modello precedentemente estrapolato sono stati ottenuti i risultati di seguito riportati:

**CR1 - Impact Sets**  
SIS: CliStarter (rinominarla in “Starter” e aggiungere l’istanziazione del tipo di parser generico)  
CIS: CliParser (deve implementare un’interfaccia “parser” specifica)

**CR2 - Impact Sets**  
SIS: SnapshotExporterFactory (aggiungere l’istanziazione di un JSONExporter)  
CIS: ParserFactory (aggiungere l’istanziazione di un JSONParser)

**CR3 - Impact Sets**  
SIS: vuoto  
CIS: vuoto