Master Degree in

Computer Science

ISCA

Identification of Self-admitted technical debts through Conversational Agent

Impact Analysis

Team Members

Alessandro Bergamo
a.bergamo2@studenti.unisa.it

Rosario Di Palma
r.dipalma22@studenti.unisa.it

Vincenzo Manserra
v.manserra@studenti.unisa.it

Reviewer

Stefano Lambiase
slambiase@unisa.it

27th March 2022

Indice

Revision History

Tabella 1: Revision History

Version	Team Member	Description	Date
0.1	Alessandro Bergamo, Rosario Di Palma, Vincenzo Manserra	Prima stesura del documento	11/04/2022
0.2	Alessandro Bergamo, Rosario Di Palma, Vincenzo Manserra	Revisione del documento ed aggiunta di altri paragrafi.	26/04/2022
0.3	Alessandro Bergamo, Rosario Di Palma, Vincenzo Manserra	Stesura della CR_1	05/05/2022
0.4	Alessandro Bergamo, Rosario Di Palma, Vincenzo Manserra	Stesura della CR_2	06/05/2022
0.5	Alessandro Bergamo, Rosario Di Palma, Vincenzo Manserra	Stesura della CR_3	07/05/2022
0.6	Alessandro Bergamo, Rosario Di Palma, Vincenzo Manserra	Stesura della CR_4 e revisione delle precedenti	09/05/2022
0.7	Alessandro Bergamo, Rosario Di Palma, Vincenzo Manserra	Stesura del sistema attuale e introduzione di ISCA	10/05/2022
0.8	Alessandro Bergamo, Rosario Di Palma, Vincenzo Manserra	Stesura dell'architettura e della matrice di dipendenza	12/05/2022
0.9	Alessandro Bergamo, Rosario Di Palma, Vincenzo Manserra	Stesura dei risultati previsti	13/05/2022
1.0	Alessandro Bergamo, Rosario Di Palma, Vincenzo Manserra	Revisione dell'intero documento e correzione dei dettagli	14/05/2022
2.0	Alessandro Bergamo, Rosario Di Palma, Vincenzo Manserra	Ultima revisione e completamento del documento	31/05/2022

1 Scopo del documento

In questo documento saranno descritti gli obiettivi del processo di manutenzione ed evoluzione sul sistema **ISCA: Identification of Self-admitted technical debts through Conversational Agent**.

Si studierà come le modifiche identificate impattano sugli artefatti del sistema esistente, infatti, per ognuna di esse è stato condotto uno studio di fattibilità apposito.

2 Panoramica del sistema attuale

Per risalire alla nascita di ISCA: Identification of Self-admitted technical debts through Conversational Agent, bisogna considerare il plugin ASATDD (Automatic Self-Admitted Technical Debt Detection tool) che nasce come tool di sostegno agli sviluppatori, ma soprattutto ai manager di progetto, per identificare la generazione di nuovi debiti tecnici semplicemente analizzando il contenuto testuale dei commit message.

La funzione di ASATDD, integralmente, è quella di eseguire un'analisi testuale dei commit message al fine di individuare e mostrare i commit che hanno potuto introdurre dei debiti tecnici auto segnalati dallo sviluppatore stesso.

2.1 Requisiti Funzionali

Le funzionalità del sistema, oggetto di questa sezione, sono le seguenti:

- **FR_1:** Il sistema deve essere in grado di recuperare i commit di una data repository
- **FR_2:** Il sistema deve essere in grado di istruire il classificatore
- **FR_3:** Il sistema deve essere in grado di eseguire l'analisi testuale dei commit message per evidenziare potenziali Self-Admitted Technical Debt
- **FR_4:** Il sistema deve segnalare i commit identificati come SATD

2.2 Design

Il sistema è stato progettato in maniera tale da essere modulare, infatti esso si compone di due layer differenti:

- **Interface Layer**: Il quale include tutte le interfacce grafiche ed in generale i boundary objects con cui l'utente dovrà interagire.
- **Application Layer**: Include tutti gli oggetti relativi alla logica di controllo ed alle entità del sistema.

Il sistema software è stato sviluppato utilizzando il linguaggio di programmazione Object Oriented Java per lo sviluppo del modello funzionale, coniugato all'apporto delle librerie Weka per l'implementazione del training del classificatore e per la classificazione delle istanze dei commit.

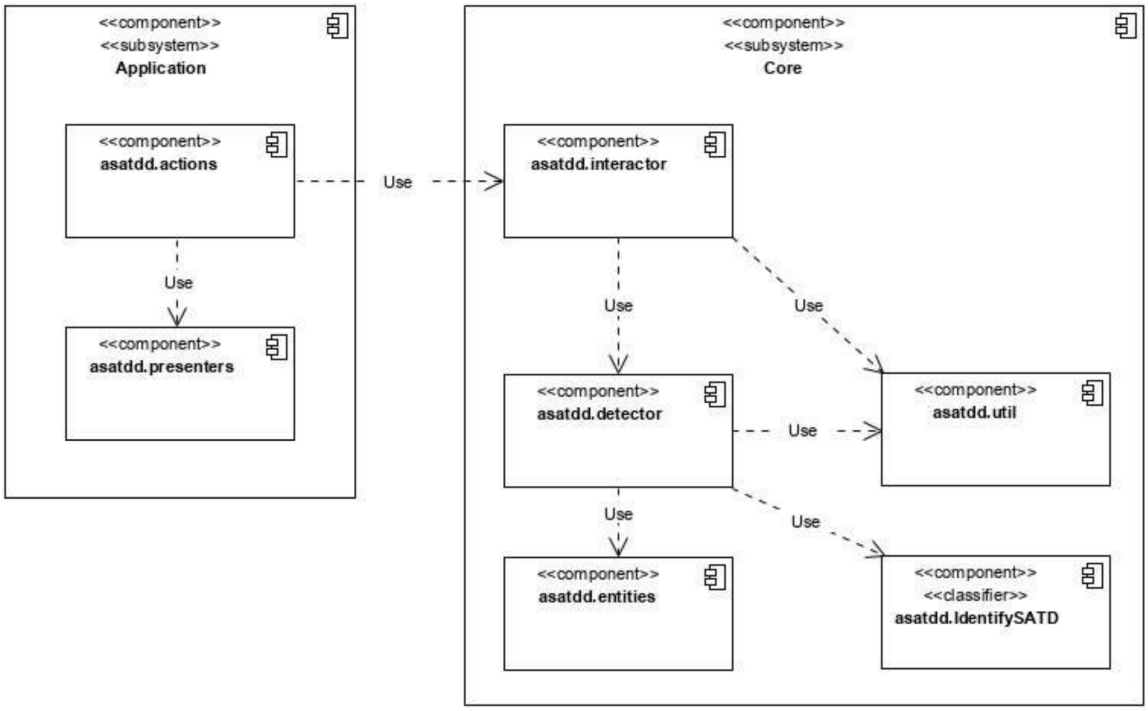
2.3 Implementazione

L'implementazione del software è stata effettuata in riferimento al modello descritto in precedenza, in particolare associando ad ogni layer un insieme di classi che sviluppano l'insieme delle funzionalità descritte in fase di analisi dei requisiti.

La **Figura 1** mostra il software primordiale su cui andremo ad effettuare le modifiche; possiamo notare che è diviso in due componenti:

- **Application**: componente che implementa l'interfaccia grafica del plugin e comunica con il componente *Core* attraverso le actions del plugin
- **Core**: componente che implementa le funzionalità dell'intero plugin, tra cui il processo di istruzione del classificatore e la logica per l'identificazione dei Self-Admitted Technical Debt

Figura 1: Architettura del plugin ASATDD



È quindi osservabile che la componente **Core** che implementa le funzionalità chiave dell'intero sistema non dipendono in alcun modo dall'altra componente, **Application**, in quanto è quest'ultima che utilizza la logica di business della componente *Core*. Possiamo quindi affermare che quest'ultima è assolutamente indipendente e slegata da vincoli di dipendeza con altre componenti.

3 Modifiche Proposte

Le change request approvate per il presente lavoro sono state:

- **CR_1:** Reingegnerizzazione del modulo di identificazione dei SATD
- **CR_2:** Sviluppo di un conversational agent
- **CR_3:** Miglioramento del classificatore
- **CR_4:** Miglioramento della gestione delle eccezioni

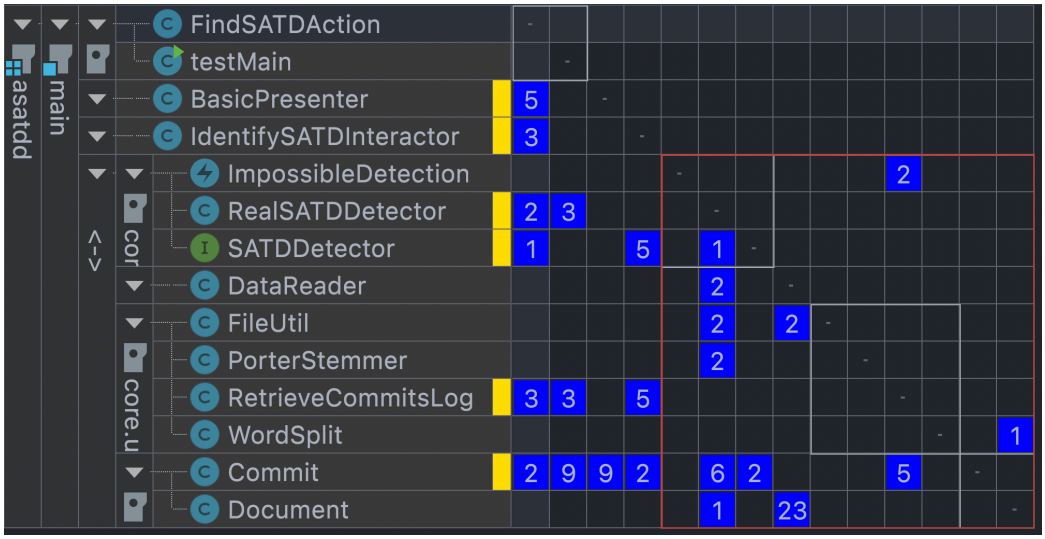
Le change request sono state presentate nell'ordine di esecuzione. Le change requests **CR_1** e **CR_4** sono state proposte per un miglioramento della parte del modulo funzionale.

Inoltre, le change request **CR_2** e **CR_3** sono state aggiunte per l'integrazione con il progetto di *Software Dependability*.

3.1 Matrice delle dipendenze

In figura è mostrata la matrice delle dipendenze del vecchio sistema.

Figura 2: Matrice delle dipendenze del vecchio sistema



3.2 Change Request – CR_1: Reingegnerizzazione del modulo di identificazione dei SATD

3.3 Descrizione della modifica

La presente change request consiste nella reingegnerizzazione del modulo di identificazione dei SATD. La riprogettazione del modulo consta dell'estrazione del vecchio modulo e del suo riutilizzo all'interno di un nuovo sistema. Il nuovo sistema software si baserà sull'utilizzo di un Conversational Agent che si avvale delle funzioni di questo modulo per identificare i SATD nei messaggi di commit.

3.3.1 Problematiche affrontate

- **Issue 1**: Per la reingegnerizzazione del modulo di identificazione dei SATD, quali funzionalità del vecchio modulo sono state estratte?
- **Resolution 1**: Per la reingegnerizzazione del modulo di identificazione dei SATD sono state riprese tutte le vecchie funzionalità facenti parte del vecchio modulo e riutilizzate all'interno del nuovo sistema.

3.3.2 Impact analysis

Analizzando la matrice delle dipendenze del vecchio sistema, in figura 2 è stato possibile notare come il modulo da re-ingegnerizzare non dipenda minimamente dall'implementazione del modulo del plugin.

Basandosi su *analisi delle dipendenze* e conoscenza dello sviluppatore si è potuto concludere che la reingegnerizzazione del modulo funzionale non dipendeva da nessun'altra classe e pertanto non richiedeva modifiche di altre classi.

Per tal motivo non è stato individuato nessun Candidate Impact Set.

3.3.3 Studio di fattiblità

Nello studio di fattibilità della change request in questione ci si è concentrati sull'analisi dei costi e sui benefici che potrebbe apportare in modo da avere una chiara valutazione sulla fattibilità della modifica.

Tabella 2: Tabella di identificazione, valutazione dei costi e descrizione

Identificazione	Valutazione	Descrizione
Reverse Engineering sul modulo funzionale	BASSA	Il team conosce largamente l'implementazione di ASATDD e possiede già dati su di esso.
Alterazioni sul modello per migliorarlo e adattarlo al nuovo sistema	BASSA	Occorrerà implementare una classe che possa semplicemente chiamare la classe interactor per interagire con l'intero modulo funzionale.
Modifiche alle classi già presenti in ASATDD	MEDIA	Occorrerà modificare alcune delle classi già presenti in ASATDD per permettere una corretta comunicazione con il nuovo modulo. Tale modifica interesserà solo import e poche linee di codice.

Tabella 3: Tabella di identificazione, valutazione dei benifici e descrizione

Identificazione	Valutazione	Motivazione
Passaggio da plugin a Conversational Agent	FORTE	Il sistema vecchio essendo un plugin era soggetto ad essere eseguito in locale su un progetto clonato manualmente, ilché risultava essere lento e poco usabile. Con il passaggio al nuovo sistema ed attraverso l'utilizzo di un conversational agent vengono meno queste problematiche, rendendo il sistema nuovo ed attuale molto più usabile.
Modularizzazione	MEDIA	Ristrutturando e ricostruendo la struttura dell'intero sistema, si aprirà la strada a future operazioni di modularizzazione che potrebbero portare alla separazione del modulo funzionale dal essere utilizzato esclusivamente come plugin per IntelliJ ed al suo utilizzo in altri ambienti.

3.3.4 Analisi Post-Implementazione

Attraverso le tecniche sopra citate si è riusciti ad estrapolare il modello in figura 4 rappresentante la logica dell'intero sistema reingegnerizzato. Nella figura 3 è possibile notare la vecchia implementazione del plugin ASATDD.

Allo stato attuale **ISCA** gestisce la comunicazione tra i due moduli, *main* e *core*, utilizzando un *bridge pattern* implementato attraverso una classe denominata come *IdentifySATDInteractor*.

Attraverso l'interactor è stato possibile rendere le due implementazioni completamente libere da ogni tipo di vincolo. A tal proposito, l'interactor ottiene le informazioni dei commit dalla classe *RetrieveCommitsLog*, utilizzando le API di GitHub e la libreria *JGit*.

Successivamente, attraverso l'utilizzo di un classificatore appositamente istruito, i commit vengono classificati per verificarne la presenza dei SATD; tutto ciò viene svolto dalla classe *RealSATDDetector* che implementa un'interfaccia *SATDDetector* che funge da *strategy pattern*, in modo da rendere più flessibile la possibilità di implementare nuovi processi d'istruzione dei classificatori.

L'implementazione della modifica è avvenuta come programmato.

3.4 Change Request – CR_2: Sviluppo di un conversational agent

3.4.1 Descrizione della modifica

Nella seguente Change Request, l'obiettivo è quello di sviluppare un conversational agent che possa sfruttare le funzionalità rese disponibili dal modulo reingegnerizzato nella prima Change Request (CR_1). Allo stato attuale il tool richiede di essere utilizzato attraverso l'IDE IntelliJ dopo averlo installato manualmente dal marketplace, eseguendolo in locale sulla propria macchina. Grazie all'utilizzo di un conversational agent vengono eliminati i vincoli citati pocanzi e rende l'intero sistema utilizzabile comodamente in un ambiente di lavoro, in questo caso SLACK, andando a migliorarne l'usabilità e le performance dell'intero tool.

La modifica non prevede di cambiare sostanzialmente il modo in cui le informazioni sui Self-Admitted Technical Debt sono rappresentate o ottenute, ma richiede solo che tali informazioni vengano rese disponibili in un ambiente esterno, staccato dall'editor principale.

3.4.2 Problematiche affrontate

Essendo la CR presentata in modo chiaro e specifico, non sono state individuate problematiche che necessitassero di analisi approfondita.

3.4.3 Impact analysis

Attualmente, ASATDD comunica con l'utente attraverso una window costruita dalle classi contenute nel package *main*. Nello specifico, la classe che si occupa di gestire la costruzione della window è *BasicPresenter*. Nel nostro piano di modifica, si prevede di eliminare la suddetta classe per rendere un canale SLACK la GUI principale del sistema da cui poterne invocare le funzionalità ed otterne la visualizzazione dei dati.

Sfruttando tecniche di analisi delle dipendenze e delle chiamate si è potuto risalire alla classe che comunica direttamente con essa, ovvero *FindSATDAction*.

A valle di queste informazioni, della conoscenza dello sviluppatore del precedente sistema e dall'analisi della matrice delle dipendenze, visibile in figura 2, si è potuto evidenziare come lo sviluppo di un conversational agent, e quindi di una nuova interfaccia grafica, non va ad impattare nessun'altra classe.

Pertanto non è stato individuato nessun Candidate Impact Set.

3.4.4 Studio di fattiblità

Nello studio di fattibilità della change request in questione ci si è concentrati sull'analisi dei costi, sui benefici e sui rischi che potrebbero scaturire in modo da avere una chiara valutazione sulla fattibilità della modifica.

Tabella 4: Tabella di identificazione, valutazione dei costi e descrizione

Identificazione	Valutazione	Descrizione
Modifica del codice per passare da window ad utilizzo di SLACK come GUI	BASSA	SLACK fornisce già una visualizzazione appropriata dei dati.

Tabella 5: Tabella di identificazione, valutazione dei benifici e descrizione

Identificazione	Valutazione	Motivazione
Maggior controllo da parte dell'utente dell'interfaccia grafica	MEDIO	Attraverso l'utilizzo di SLACK come GUI è possibile avere un'esperienza di visualizzazione dei dati migliorata, soprattutto grazie alla disponibilità dei log già disponibili all'interno del canale SLACK in cui è installato il sistema.

Tabella 6: Tabella di identificazione, valutazione dei rischi e descrizione

Identificazione	Valutazione	Motivazione
Introduzione di bug grafici dovuti all'utilizzo del Conversational Agent come GUI	MEDIO	Ricevendo i dati tramite delle richieste e risposte HTTP RESTFUL la visualizzazione dei dati potrebbe essere confusionaria a causa del fatto che i dati vengono inviati e ricevuti tramite file JSON. Per tal motivo, se dovessero esserci problemi, verrà effettuata una manipolazione dei dati per fornirne una visualizzazione dettagliata, precisa e ben comprensibile.

3.4.5 Analisi Post-Implementazione

Attraverso le tecniche sopra citate si è riusciti a costruire la nuova logica di presentazione correttamente. La nuova visualizzazione è visibile in figura 3.5.

3.5 Change Request – CR_3: Miglioramento del classificatore

3.5.1 Descrizione della modifica

Nella seguente Change Request l'obiettivo è quello di richiedere un'evoluzione del classificatore attraverso il miglioramento del processo di istruzione. In conformità con gli standard di qualità che sono stati stabiliti in precedenza, è stato richiesto quindi il miglioramento della precisione del classificatore.

3.5.2 Problematiche affrontate

- **Issues 1**: In che modo avverrà il miglioramento del Classificatore?
- **Resolution 1**: Il miglioramento del classificatore avverrà attraverso tre step:
  1. Rivisitando il dataset di addestramento già presente
  2. Rivisitando la tipologia di classificatore (albero, funzionale, testuale, etc.)
  3. Rivisitando la valutazione degli attributi e la selezione delle feature

Tale cambiamento mira ad ottenere un miglioramento della qualità e dell'affidabilità dell'intero sistema. In figura sono mostrati i parametri del classificatore che veniva utilizzato in precedenza.

Figura 3: Metriche classificatore NaiveBayesMultinomial

```
=== CONFUSION MATRIX FOR FOLD 5/5 ===
   a   b   <-- classified as
 915   6 |   a = negative
  23  26 |   b = positive

Correct % = 97.01030927835052
Incorrect % = 2.9896907216494846
Precision = 0.8125
Recall = 0.5306122448979592
fMeasure = 0.6419753086419753
Error Rate = 0.029896907216494847
```

3.5.3 Impact analysis

Essendo che il miglioramento del classifcatore implica la rivisitazione del dataset di addestramento e di tutto il processo di istruzione del classificatore stesso, queste modifiche non vanno ad impattare su nessun'altra classe.

A tal proposito non è stato individuato nessun Candidate Impact Set.

3.5.4 Studio di fattibilità

Nello studio di fattibilità della change request in questione ci si è concentrati sull'analisi dei costi, sui benefici e sui rischi che potrebbero scaturire in modo da avere una chiara valutazione sulla fattibilità della modifica.

Tabella 7: Tabella di identificazione, valutazione dei costi e descrizione

Identificazione	Valutazione	Descrizione
Modifica del processo di istruzione del classificatore	BASSA	Il team conosce largamente l'implementazione del codice d'istruzione del classificatore e possiede già dati su di esso.
Modifica del dataset per l'istruzione del classificatore	BASSA	Il team conosce largamente l'intero dataset e possiede già dati su di esso.

Tabella 8: Tabella di identificazione, valutazione dei benifici e descrizione

Identificazione	Valutazione	Motivazione
Maggior precisione sull'identificazione dei Self-Admitted Technical Debt	MEDIO	Attraverso il miglioramento del processo di istruzione del classificatore si mira ad incrementare la precisione con cui quest'ultimo riesce ad identificare i SATD.

Tabella 9: Tabella di identificazione, valutazione dei rischi e descrizione

Identificazione	Valutazione	Motivazione
Peggioramento della precisione con cui il classificatore individua i Self-Admitted Technical Debt	MEDIO	Durante l'implementazione delle nuove tecniche di istruzione del classificatore potrebbe accadere che le sue prestazioni, espresse in termini di precisione, calino drasticamente fornendo così un'implementazioe non più affidabile.

3.5.5 Analisi Post-Implementazione

Attraverso le tecniche sopra citate si è riusciti a costruire la nuova logica di istruzione del classificatore.

Durante il processo di miglioramento si è deciso di voler puntare più alla *precision* che alla *recall*. Difatti si è andati a valutare le metriche in base a questa caratteristica. Inoltre, si è scelto di voler preferire i falsi positivi piuttosto che i falsi negativi. Si è notato come la presenza di falsi positivi poteva impattare sul numero di Self-Admitted Technical Debt segnalati, ma non sull'efficienza del classificatore.

Nelle successive figure vengono mostrati tutti i parametri relativi ai tentativi svolti durante il miglioramento del classificatore.

La figura 4 mostra il classificataore di tipo *tree* J48 che presenta un'elevata percentuale di falsi positivi rispetto alle figure seguenti. Ciò ha fatto si che questa tipologia di classificatore venisse scartato soprattutto dovuto al fatto che la *precision* è pressocché allo stesso livello degli altri. Discorso non analogo per la *recall*, molto più bassa rispetto agli altri.

Figura 4: Metriche classificatore J48

```
=== CONFUSION MATRIX FOR FOLD 5/5 ===
   a   b   <-- classified as
 920   1 |   a = negative
  37  12 |   b = positive

Correct % = 96.08247422680412
Incorrect % = 3.917525773195876
Precision = 0.9230769230769231
Recall = 0.24489795918367346
fMeasure = 0.3870967741935484
Error Rate = 0.03917525773195876
```

Di seguito, nella figura 5 troviamo il classificatore *rule* DecisionTable che presenta delle buone metriche a paragone di tutti gli altri.

L'unico svantaggio di questo classificatore è che tende ad identificare più falsi negativi di quanti ne identificano gli altri classificatori, andando così in contrasto con i nostri parametri di valutazione delle metriche di ogni classificatore.

Figura 5: Metriche classificatore DecisionTable

```
=== CONFUSION MATRIX FOR FOLD 5/5 ===
   a   b   <-- classified as
 919   2 |   a = negative
  26  23 |   b = positive

Correct % = 97.11340206185567
Incorrect % = 2.88659793814433
Precision = 0.92
Recall = 0.46938775510204084
fMeasure = 0.6216216216216216
Error Rate = 0.0288659793814433
```

Di seguito, nella figura 6 troviamo il classificatore *lazy* IBk che presenta delle buone metriche a paragone di tutti gli altri, quasi al pari delle metriche offerte dalla tipologia di classificatore scelto.

L'unico svantaggio di questa tipologia è che si rivela molto lento nella sua istruzione, rallentando così l'esecuzione dell'intero sistema.

Figura 6: Metriche classificatore IBk

```
=== CONFUSION MATRIX FOR FOLD 5/5 ===
   a   b   <-- classified as
 919   2 |   a = negative
  25  24 |   b = positive

Correct % = 97.21649484536083
Incorrect % = 2.783505154639175
Precision = 0.9230769230769231
Recall = 0.4897959183673469
fMeasure = 0.64
Error Rate = 0.027835051546391754
```

Tuttavia si è deciso di utilizzare un classificatore di tipo *tree* denominato come REPTree. Si è notato che questa tipologia ha le migliori metriche tra i classificatori sopra citati.

Possiamo notare che il numero di falsi negativi individuati è davvero minimo e che la *precision* è di gran lunga la migliore tra quelli provati in precedenza, andando anche a considerare la *recall* che è mediamente alta rispetto ai parametri dei precedenti classificatori; tutti questi motivi ci hanno spinto a considerare questa tipologia di classificatore come il migliore tra quelli provati.

Figura 7: Metriche classificatore REPTree

```
=== CONFUSION MATRIX FOR FOLD 5/5 ===
   a   b   <-- classified as
 920   1 |   a = negative
  27  22 |   b = positive

Correct % = 97.11340206185567
Incorrect % = 2.88659793814433
Precision = 0.9565217391304348
Recall = 0.4489795918367347
fMeasure = 0.6111111111111112
Error Rate = 0.0288659793814433
```

Proseguendo, si è deciso di cambiare il modo con cui venivano selezionate le *feature* andandole a selezionare con un criterio differente rispetto a quello che veniva utilizzato.

Infatti si è deciso di utilizzare il criterio di selezione *BestFirst* in cui vengono scelte le feature attraverso una ricerca nello spazio degli attributi in maniera *greedy hillclimbing* unito ad una funzione di backtracking che permette di considerare più volte tutti gli attributi, anche quelli già individuati, come potenziali feature.

3.6 Change Request – CR_4: Miglioramento della gestione delle eccezioni

3.6.1 Descrizione della modifica

Nella seguente Change Request si richiede l'aggiunta o la modifica delle eccezioni in riferimento agli errori che possono verificarsi durante l'esecuzione del sistema.

Allo stato attuale, quando si verifica un errore di sistema non viene segnalata la causa che ha portato all'errore.

Per tale motivo, vogliamo fornire una chiara motivazione della causa dell'errore in modo che l'utente possa ovviare al problema, più specificamente evitando errori che dipendano dall'utente stesso.

3.6.2 Problematiche affrontate

Essendo la CR presentata in modo chiaro e specifico, non sono state individuate problematiche che necessitassero di analisi approfondita.

3.6.3 Impact analysis

Utilizzando il nuovo sistema è stata svolta un analisi statica del codice sorgente per analizzare le classi più critiche e che avrebbero potuto generare errori durante l'esecuzione.

In supporto è stata anche utilizzata la matrice delle dipendenze, in figura 2, per poter avere un quadro più chiaro delle classi che avevano più interazioni tra loro.

Tra queste sono state identificate due classi:

- *RetrieveCommitsLog*: classe che si occupa di recupare le informazioni riguardanti i commit di una determinata repository
- *RealSATDDetector*: classe che si occupa dell'istruzione del classificatore e della classificazione dei commit message

Per tale motivo, tali classi sono state identificate nello Starting Impact Set illustrato nella tabella 10. Dopo varie valutazioni, abbiamo constatato che lo Starting Impact Set coincide con il Candidate Impact Set.

L'impatto della modifica verrà valutato utilizzando tre categorie:

- **FORTE**: se saranno necessarie pesanti modifiche nell'artefatto o se l'artefatto dovrà essere completamente sostituito;
- **MEDIO**: se saranno necessarie sostanziali modifiche all'artefatto, non facendo cambiare però la sua struttura in maniera eccessiva;
- **DEBOLE**: se saranno necessarie solo modifiche marginali.

Tabella 10: Candidate Impact Set

Artefatto	Impatto	Descrizione
RetrieveCommitsLog.java	MEDIO	Il team conosce largamente le classi ed i metodi che la classe implementa, pertanto sarà pressocché semplice far si che vengano lanciate le corrette eccezioni nelle sezioni critiche della classe.
RealSATDDetector.java	MEDIO	Il team conosce largamente le classi ed i metodi che la classe implementa, pertanto sarà pressocché semplice far si che vengano lanciate le corrette eccezioni nelle sezioni critiche della classe.

3.6.4 Studio di fattibilità

Nello studio di fattibilità della change request in questione ci si è concentrati sull'analisi dei costi e sui vantaggi che potrebbero scaturire in modo da avere una chiara valutazione sulla fattibilità della modifica.

Tabella 11: Tabella di identificazione, valutazione dei costi e descrizione

Identificazione	Valutazione	Descrizione
Aggiunta di nuove eccezioni al sistema	MEDIA	Il team conosce largamente le classi ed i metodi che il sistema implementa, pertanto sarà relativemnte semplice far si che vengano lanciate le corrette eccezioni qualora si verificasse un errore.

Tabella 12: Tabella di identificazione, valutazione dei benifici e descrizione

Identificazione	Valutazione	Motivazione
Maggior chiarezza quando il sistema genera un errore	MEDIO	Attraverso l'aggiunta di nuove eccezioni il sistema diventerà molto più usabile soprattutto perché l'utente potrà rendersi conto se l'errore generato dal sistema è causato dall'utente stesso.

3.6.5 Analisi Post-Implementazione

Attraverso le tecniche sopra citate si è riusciti ad aggiungere nuove eccezioni che permettono di fornire una corretta spiegazione dell'errore che si è verificato.

3.7 Metriche

Di seguito, vengono riportate le principali metriche dell'information retrieval utilizzate per la valutazione dell'impact analysis effettuata.

$$Recall = \frac{|CIS \cap AIS|}{|AIS|} \qquad Precision = \frac{|CIS \cap AIS|}{|CIS|} \qquad F - measure = 2 * \frac{Precision * Recall}{Precision + Recall}\%$$

Per quanto riguarda la **CR_1**, **CR_2**, **CR_3** non sono stati individuati dei Candidate Impact Set in quanto la loro realizzazione non impattava su alcuna classe del sistema, pertanto non verranno calcolate le metriche.

Per quanto riguarda la **CR_4** è stato individuato un Candidate Impact Set (CIS) di dimensione *due* che coincide esattamente con l'Actual Impact Set (AIS).

A tal proposito, sono state calcolate le seguenti metriche:

- **Precision**:

$$\frac{|CIS| \cap |AIS|}{|CIS|} = \frac{|2|}{|2|} = 100\%$$

- **Recall**:

$$\frac{|CIS| \cap |AIS|}{|AIS|} = \frac{|2|}{|2|} = 100\%$$

- **f-Measure**:

$$2 * \frac{1+1}{1+1}\% = 100\%$$

4 Sistema attuale - ISCA

Le modifiche sono state implementate correttamente ed hanno permesso ad ISCA di godere sia di miglioramenti dal punto di vista dei propri attributi interni ed esterni, sia dal punto di vista delle funzionalità.

In figura 8 è possibile vedere la nuova architettura di ISCA.

4.1 Requisiti Funzionali

Le funzionalità del nuovo sistema, oggetto di questa sezione, sono le seguenti:

- **FR_1:** Il conversational agent deve essere utilizzabile tramite SLACK
- **FR_2:** Il conversational agent deve mostrare all'utente i Self-Admitted Technical Debt di una qualsiasi repository
- **FR_3:** Il conversational agent deve essere in grado di mostrare la lista dei commit completa di una qualsiasi repository

4.2 Design

Il sistema è stato progettato in maniera tale da essere modulare, infatti esso si compone di tre layer differenti:

- **Interface layer**: il quale include l'interfaccia grafica utente
- **Application layer**: il quale include la parte di gestione delle richieste
- **Core layer**: il quale include tutta la logica core del sistema

È stata effettuata questa scelta considerando anche future modifiche del sistema, rendendolo più manutenibile e flessibile a nuovi aggiornamenti cercando di non far dipendere nessun modulo dall'implementazione dell'altro, rispettando così il principio di *high cohesion and low coupling.*

4.3 Implementazione

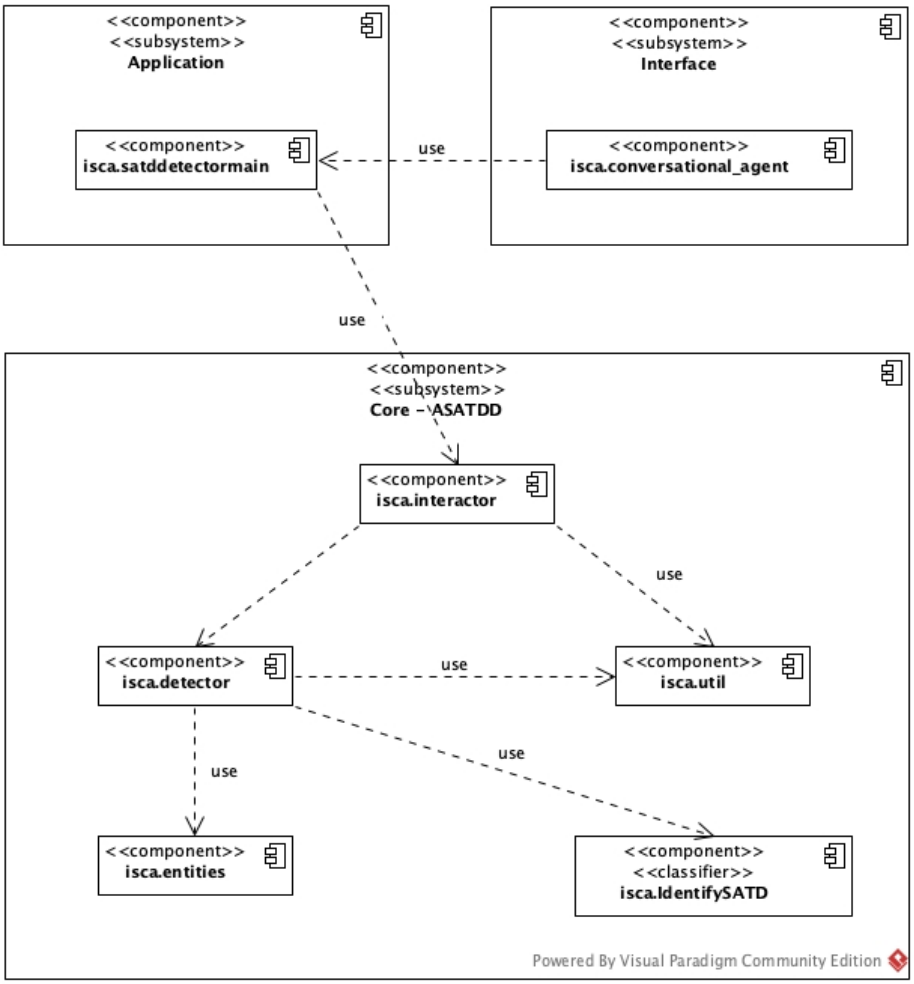
Come possiamo notare dalla figura 7 viene illustrato il component diagram del nuovo software su cui sono state effettuate le modifiche diviso, come indicato nella sezione precedente, in tre componenti:

- **Interface layer:** componente che rappresenta la GUI fornita da SLACK e i "*slash commands* da poter utilizzare per sfruttare le funzionalità messe a disposizione tramite l'Application layer
- **Application layer:** componente che rappresenta la classe che fornisce i servizi sottoforma di richieste e risposte HTTP sfruttando il framework SpringBoot, in modo da ricevere dati dal Core layer ed inviarli all'Interface layer
- **Core layer:** componente reingegnerizzata che implementa le funzionalità dell'intero sistema, tra cui il processo di istruzione del classificatore e la logica per l'identificazione dei Self-Admitted Technical Debt. Tale layer restituisce i commit identificati al layer sovrastante, l'Application layer

Il nuovo sistema software è stato sviluppato utilizzando:

- **Java** come linguaggio di programmazione Object Oriented per lo sviluppo del modello funzionale, in congiunto con l'utilizzo delle librerie Weka per l'implementazione del training del classificatore e della classificazione delle istanze
- **Python** per lo sviluppo del Conversational Agent

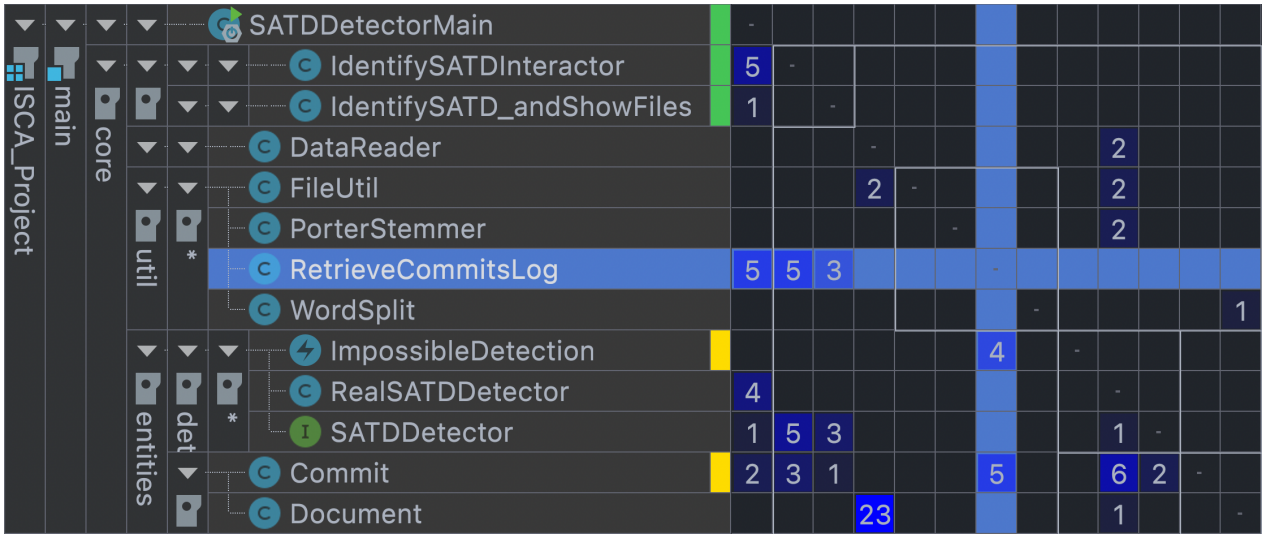
Figura 8: Architettura della nuova implementazione - ISCA



4.4 Matrice delle dipendenze

Di seguito è mostrata la matrice delle dipendenze del nuovo sistema in seguito alle modifiche effettuate.

Figura 9: Matrice delle dipendenze



5 Risultati previsti

In conclusione, le modifiche qui citate sono state approvate sulla base degli studi e delle analisi proposte in questo documento.

Le previsioni fatte si sono rivelate corrette, pertanto l'implementazione dei cambiamenti non ha prodotto risultati inattesi.