Master Degree in

Computer Science

ISCA

Identification of Self-admitted technical debts through Conversational Agent

Test Plan & Specification - ISCA

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

25 Maggio 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	25/05/2022
1.0	Alessandro Bergamo, Rosario Di Palma, Vincenzo Manserra	Stesura finale del documento	11/06/2022

1 Introduzione

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

Si andrà a pianificare e specificare come verranno strutturati ed eseguiti i casi di test rivolti ad ogni classe aggiunta nel nuovo sistema, rispetto al precedente; in aggiunta verrà pianificato e specificato il test metamorfico che servirà per verificare che il sistema dopo esser stato reingegnerizzato produce la stessa tipologia di output considerando la stessa tipologia di input.

Un test case è un insieme di input e di risultati attesi che servono a testare una componente per individuare comportamenti diversi da quelli attesi, cioè i failure.

2 Pianificazione e specificazione casi di test

Nella prossima sezione andremo ad esplicitare tutti i criteri dei casi di test e come sono stati pianificati.

Inoltre, in dettaglio saranno esplicitate le specifiche di ogni singolo caso di test.

2.1 Requisiti funzionali

Di seguito vengono esplicitati i requisiti funzionali dell'intero sistema ISCA compresi quelli relativi al vecchio sistema da cui è stato reingegnerizzato il modulo funzionale.

- **FR_1:** Il sistema deve essere utilizzabile tramite SLACK
- **FR_2:** Il sistema deve permettere all'utente di interagire con un Conversational Agent
- **FR_3:** Il sistema deve essere in grado di recuperare i commit di una data repository
- **FR_4:** Il sistema deve essere in grado di istruire il classificatore
- **FR_5:** Il sistema deve essere in grado di eseguire l'analisi testuale dei commit message per evidenziare potenziali Self-Admitted Technical Debt
- **FR_6:** Il sistema deve segnalare i commit identificati come Self-Admitted Technical Debt

Di questi requisiti funzionali ne verranno testati solamente una parte in quanto FR_3, FR_4, FR_5 appartengono al modulo reingnegnerizzato e pertanto verranno considerate solamente nel **testing del vecchio modulo (*metamorphic test*)**.

Per quanto riguarda il primo requisito funzionale FR_1 non si pianificherà nessun test case in quanto è un requisito per l'utilizzo dell'intero sistema piuttosto che un requisito implementativo.

Pertanto verranno testati solamente due requisiti funzionali: FR_2 e FR_6 che verranno interamente testati con Selenium IDE e SLACK.

Tabella 2: Pianificazione e specifica dei test case in riferimento ai requisiti funzionali

Test Case	Identificativo	Breve descrizione
TC_1	FR_2	L'utente potrà richiedere al Conversational Agent di identificare i SATD di una determinata repository
TC_2	FR_6	Una volta identificati, i SATD verranno mostrati a video all'utente tramite il Conversational Agent

Unito al testing di questi due requisiti verranno, inoltre, testate le componenti aggiunte al sistema per favorire l'integrazione del modulo funzionale reingegnerizzato alla nuova UI (*User Interface*) fornita dal Conversational Agent, in questo caso tramite SLACK.

Tabella 3: Pianificazione e specifica dei test case in riferimento alle nuove componenti aggiunte

Test Case	**Artefatto**	**Breve descrizione**
TC_3	SATDDetectorMain.java	Artefatto che si occupa di ricevere richieste dal conversational agent e risponde con una lista di commit identificati come SATD. Verrà testato il metodo che gestisce tali richieste andando a considerare il parametro di input *repository_url*
TC_4	RetrieveCommitsLog.java	Artefatto che si occupa di recuperare i commit di una determinata repository. Verrà testato il caso in cui l'url della repository non corrisponde ad una repository reale, ma è comunque considerato valido (in riferimento al TC_3)
TC_5	RealSATDDetector.java	Artefatto che si occupa dell'identificazione dei SATD. Verrà testato il caso in cui nella repository data ci sono troppi pochi commit per effettuarne una classificazione

3 Criteri pass/failed

Per ogni failure riscontrato durante l'esecuzione dei test case, sarà individuato il relativo fault e si procederà alla sua correzione. Successivamente, sarà reiterata la fase di testing per verificare che le modifiche apportate non abbiano avuto un impatto dannoso sulle altre componenti del sistema.

4 Approccio

La fase di testing si articolerà in due momenti:

- Inizialmente verrà effettuato il testing di unità delle singole componenti che sono state aggiunte al precedente modulo funzionale o che sono state modificate in seguito alle modifiche previste dalla CR_4 nella quale viene migliorata la gestione delle eccezioni all'interno del sistema; in questo modo, sarà possibile andare ad effettuare degli interventi di manutenzione, in caso fossero necessari, e verificare che le modifiche apportate non abbiano intaccato le funzionalità iniziali;
- Al termine, sarà effettuata un'ulteriore sessione di testing, durante la quale verrà effettuato il testing di regressione, attraverso l'approccio del metamorphic testing; infine, verranno svolte attività di testing di integrazione, per verificare, rispettivamente, il funzionamento delle singole componenti del sistema e l'integrazione dei singoli sottosistemi fino ad arrivare al testing del sistema completo.

5 Testing di unità

Il testing di unità avrà l'obiettivo di testare la correttezza delle singole componenti aggiunte al sistema e sarà effettuato utilizzando l'approccio black-box.

5.1 Test Case

La test suite del testing di unità è definita adottando un criterio di copertura che prevede la definizione dei casi di test mediante **category partition**; per ciascuna funzionalità da testare saranno individuati i parametri di input e gli oggetti dell'ambiente; per ciascun parametro, saranno individuate le categorie di cui si compone e, per ciascuna categoria, saranno individuate tutte le possibili scelte.

Questo approccio permetterà di individuare le relazioni di incompatibilità, per le quali non saranno realizzati test case.

I test case che andremo a considerare in questa sezione sono stati individuati ed esplicitati nelle tabelle 2 - 3.

5.2 TC_1 - L'utente potrà richiedere al Conversational Agent di identificare i SATD di una determinata repository

Parametro: Comando richiesto	
CATEGORIE	**SCELTE**
Esiste[ec]	1. Il comando non esiste 2. Il comando esiste

Parametro: Stringa	
CATEGORIE	**SCELTE**
Inserita[is]	1. La stringa non è inserita 2. La stringa è stata inserita

IDENTIFICATIVO	COMBINAZIONE	ESITO
TC_1_1	**ec1is1**	**Non è un comando valido**
TC_1_2	**ec2is1**	**Comando non eseguito con successo**
TC_1_3	**ec1is2**	**Non è un comando valido**
TC_1_4	**ec2is2**	**Comando eseguito con successo**

5.3 TC_2 - Una volta identificati, i SATD verranno mostrati a video all'utente tramite il Conversational Agent

Parametro: SATD	
CATEGORIE	**SCELTE**
Esiste[es]	1. Non sono stati trovati SATD 2. È stato trovato almeno un SATD

IDENTIFICATIVO	COMBINAZIONE	ESITO
TC_2_1	**es1**	**Non sono stati rilevati SATD**
TC_2_2	**es2**	**Comando eseguito con successo**

5.4 TC_3 - SATDDetectorMain.java - Lancio eccezione WrongRepositoryLink.java

Parametro: repository_url	
CATEGORIE	**SCELTE**
Corrisponde ad una repository[cp]	1. Il parametro repository_url non corrisponde ad una repository 2. Il parametro repository_url corrisponde ad una repository

IDENTIFICATIVO	COMBINAZIONE	ESITO
TC_3_1	**cp1**	**L'URL non corrisponde ad una repository valida**
TC_3_2	**cp2**	**Comando eseguito con successo**

5.5 TC_4 - RetrieveCommitsLog.java - Lancio eccezione RepositoryNotFound.java

Parametro: repository_url	
CATEGORIE	**SCELTE**
Esiste[er]	1. La repository non esiste 2. La repository esiste

IDENTIFICATIVO	COMBINAZIONE	ESITO
TC_4_1	**er1**	**Repository non trovata**
TC_4_2	**er2**	**Comando eseguito con successo**

5.6 TC_5 - RealSATDDetector.java - Lancio eccezione NotEnoughCommits.java

Parametro: List⟨Commit⟩	
CATEGORIE	**SCELTE**
Sufficiente[ns]	1. Numero di commit non sufficiente 2. Numero di commit sufficiente

IDENTIFICATIVO	COMBINAZIONE	ESITO
TC_5_1	**ns1**	**Il numero di commit è inferiore al limite**
TC_5_2	**ns2**	**Comando eseguito con successo**

6 Testing di regressione del modulo

In questa sezione si andrà ad esplicitare i casi di test ripresi dal sistema precedentemente realizzato, considerando il formato dell'input e dell'output.

6.1 TC_6: Il sistema deve essere in grado di recuperare i commit di una data repository

Parametro: PATH della repository	
CATEGORIE	**SCELTE**
Esiste[er]	1. Path non valido 2. Path valido

IDENTIFICATIVO	COMBINAZIONE	ESITO
TC_6_1	**er1**	**Path non valido**
TC_6_2	**er2**	**Comando eseguito con successo** • **return type: List⟨Commit⟩**

6.2 TC_7: Il sistema deve essere in grado di istruire il classificatore

Parametro: TrainingSet	
CATEGORIE	**SCELTE**
Ben formato [tb]	1. Il TrainingSet non esiste 2. Il TrainingSet è ben formato

IDENTIFICATIVO	COMBINAZIONE	ESITO
TC_7_1	**tb1**	**Il TrainingSet non esiste**
TC_7_2	**tb2**	**Comando eseguito con successo** • **return type: Classifier**

6.3 TC_8: Il sistema deve poter analizzare testualmente i commit message e evidenziare i possibili Self-Admitted Technical Debt

Parametro: List⟨Commit⟩	
CATEGORIE	**SCELTE**
Vuota[lv]	1. La lista è vuota 2. La lista ha almeno un commit

IDENTIFICATIVO	COMBINAZIONE	ESITO
TC_8_1	**lv1**	**Non ci sono commit da classificare**
TC_8_2	**lv2**	**Comando eseguito con successo** • **return type: List⟨Commit⟩**

7 Testing di integrazione

Nel sistema abbiamo la presenza di più moduli,per tale motivo, il test di integrazione del sistema verifica una combinazione di singoli moduli nel loro insieme.

La strategia adottata per il testing di integrazione sarà di tipo Big Bang, le componenti verranno prima testate individualmente, tramite *Unit Testing* e poi testate insieme come se fossero un singolo modulo.

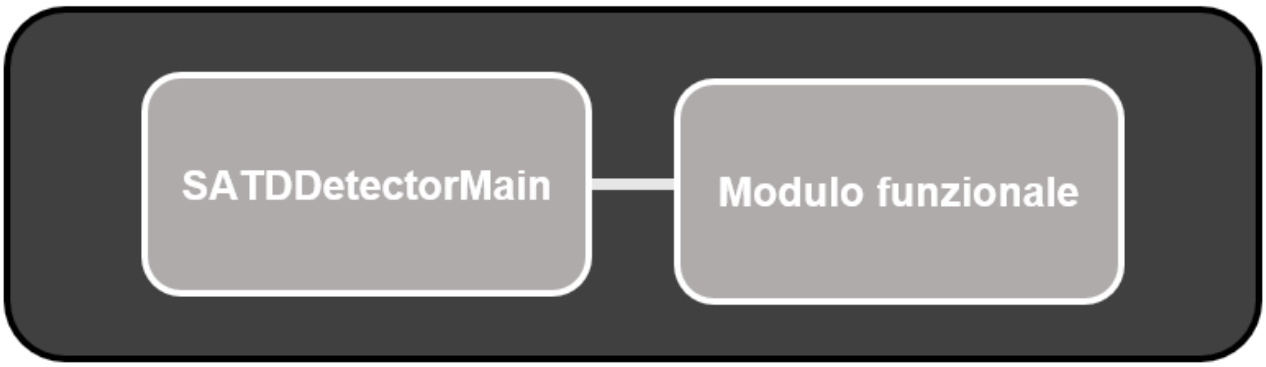
Avendo un modulo funzionale reingegnerizzato, abbiamo testato l'integrazione tra il modulo e la nuova classe che si interfaccia con esso per invocarne le funzionalità.

7.1 Componenti da testare

I sottosistemi sottoposti al testing di integrazione sono:

- **Modulo funzionale**: contentente le funzionalità per l'identificazione dei Self-Admitted Technical Debt
- **SATDDetectorMain**: contenente i metod che si occupano di ricevere e rispondere alle richieste provenienti dal Conversational Agent (considerato nel testing di sistema).

Figura 1: Sottosistemi da integrare



7.2 Criteri pass/fail

Per ogni failure riscontrato durante l'esecuzione, sarà individuato il relativo fault e si procederà alla sua correzione.

Successivamente, sarà reiterata la fase di testing per verificare che le modifiche apportate non abbiano avuto un impatto dannoso sulle altre componenti del sistema.

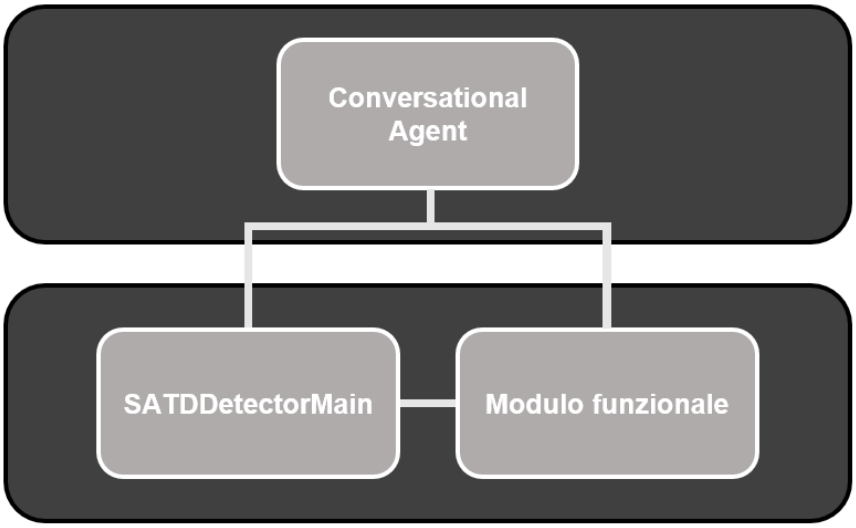
8 Testing di sistema

Il testing di sistema avrà l'obiettivo di testare l'intero sistema completo di ogni modifica.

Il testing di sistema controlla il comportamento dell'intero sistema nel suo complesso. Inoltre, aiuterà a valutare la conformità del sistema ai requisiti specificati e controlla il sistema completamente integrato per verificare che il sistema soddisfi i requisiti specificati.

Per rendere il tutto automatizzato si è scelto di effettuare tale testing di sistema utilizzando *Selenium IDE*, eseguendolo sull'interfaccia WEB di SLACK, in modo da avere un testing del sistema completo, compreso di UI.

Figura 2: Intero sistema da testare



8.1 Criteri pass/fail

Per ogni failure riscontrato durante l'esecuzione, sarà individuato il relativo fault e si procederà alla sua correzione.

Successivamente, sarà reiterata la fase di testing dell'intero sistema per verificare che le modifiche apportate abbiano avuto un impatto positivo sull'intero funzionamento del sistema.