UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Laurea Magistrale in Informatica

Corso di

Ingegneria, Gestione ed Evoluzione del Software

Test Plan Document

DOCENTE

Prof. Andrea De Lucia

AUTORI

Alfonso Cannavale

Matricola: 0522501597

Davide La Gamba

Matricola: 0522501464

Kevin Pacifico

Matricola: 0522501501

LINK REPOSITORY - BRANCH "DEV"

CADOCS: `https://github.com/alfcan/CADOCS/tree/dev`

CADOCS_NLU: `https://github.com/alfcan/CADOCS_NLU_Model/tree/dev`

csDetector: `https://github.com/alfcan/csDetector/tree/dev`

Anno Accademico 2022-2023

Indice

1 Introduzione **1**

2 Funzionalità da testare **2**

3 Criteri di adeguatezza **3**

4 Strategie **4**

4.1 Testing di Unità 4

4.2 Testing di Integrazione 5

4.3 Testing di Sistema 6

4.4 Testing di Regressione 6

4.5 Ispezione del Codice 6

5 Test Case **7**

5.1 Test Case per RF get_smells 9

5.2 Test Case per RF get_smells_date 10

5.3 Test Case per RF report 11

5.4 Test Case per RF info 11

5.5 Test Suite 11

Capitolo 1

Introduzione

CADOCS è un chatbot disponibile per la piattaforma Slack per il supporto all'individuazione di community smells, come già descritto nel documento "Pre-Maintenance Report" al quale si fa riferimento.

Al momento, il modulo CADOCS non dispone di una test suite per le sue funzionalità. Pertanto, è fondamentale stabilire un Test Plan che consenta la corretta implementazione delle nuove funzionalità utilizzando i test per evitare l'introduzione di nuovi problemi o regressioni durante il processo di evoluzione del chatbot. Quindi, sarà creata una test suite di partenza che sarà poi ampliata all'occorrenza con l'introduzione di test per le nuove funzionalità.

L'obiettivo è di migliorare l'applicazione attraverso l'implementazione delle Change Request, per il quale verrà utilizzato il Test Plan per migliorare la qualità del processo complessivo. In particolare, verranno identificate le funzionalità dell'applicazione che devono essere testate, definendo le strategie di testing da adottare, stabilendo i criteri di adeguatezza e creando i casi di test necessari.

Capitolo 2

Funzionalità da testare

A seguito di una fase di reverse engineering sono state individuate le seguenti funzionalità chiave, le quali rappresentano il core del sistema:

- **get_smells**: funzionalità che permette di analizzare una repository fornita in input, restituendo i relativi community smells identificati dal tool *csDetector*. Questa funzione consente agli utenti di ottenere un resoconto dettagliato sui community smells presenti nel progetto analizzato.
- **get_smells_date**: funzionalità che permette di analizzare una repository fornita in input da una data specificata in poi, restituendo i relativi community smells identificati dal tool *csDetector*. In questo modo, gli utenti possono ottenere informazioni dettagliate riguardanti i community smells emersi dopo una data precisata.
- **report**: funzionalità che permette all'utente di visualizzare l'ultima analisi effettuata durante la conversazione con il chatbot CADOCS.
- **info**: funzionalità che permette all'utente di visualizzare le informazioni relative ai community smells che il chatbot può identificare.

I requisiti specificati dovranno essere testati sia per le richieste formulate in lingua inglese sia in lingua italiana, utilizzando la piattaforma Slack e la piattaforma Discord.

Le funzionalità elencate costituiscono le interazioni che gli utenti possono avere con il chatbot CADOCS. Durante il processo di testing, ci si concentrerà sulla verifica della corretta implementazione di queste funzionalità.

Capitolo 3

Criteri di adeguatezza

Un test viene considerato **passato** quando il comportamento ottenuto è uguale al comportamento atteso dall'oracolo. Nel caso in cui il comportamento ottenuto sia diverso dall'oracolo allora il test viene considerato **fallito**, ciò significa che è presente una **failure** nel sistema. Saranno testati tutti i requisiti indicati nel Capitolo 2, mentre per i test di unità e di integrazione il requisito minimo è il raggiungimento del 75% di branch coverage (questo criterio è riferito ai test strutturali white-box), che dovrà essere rispettato anche durante l'esecuzione dei test di regressione.

Capitolo 4

Strategie

Al fine di soddisfare i criteri di adeguatezza del testing, in questo capitolo definiamo delle opportune strategie di testing ed i tool che verranno utilizzati. Verranno effettuati diversi livelli di testing, in particolare testing a livello di unità, di integrazione, di sistema, testing di regressione e pratiche di ispezione del codice per individuare possibili errori introdotti.

4.1 Testing di Unità

Il testing di unità consiste nel testare le singole componenti. Ciò verrà effettuato tramite una strategia di tipo **white-box** con l'utilizzo delle librerie Python: *pytest* e *unittest*. Saranno testati i metodi delle classi che compongono il sistema e sarà creata una classe di test per ogni classe del sistema da testare. Per quanto riguarda le funzionalità già implementate dal sistema CADOCS, saranno aggiunti dei casi di test basati sulle classi e sui metodi presenti, in modo da poterli riutilizzare in fase di test di regressione. Per testare il modello di Natural Language Understanding, si utilizzerà la tecnica dell'Input Testing, in modo da generare nuovi dati di input per il modello di Machine Learning tramite tool automatici, in modo da valutarne le prestazioni. Per misurare la branch coverage utilizzeremo il tool *pytest-cov*.

4.2 Testing di Integrazione

Il testing di integrazione verrà effettuato tramite l'utilizzo del framework **pytest**. La fase di testing di integrazione sarà effettuata seguendo una strategia **bottom-up**, testando l'integrazione solo tra le classi presenti nel sistema *CADOCS*.

Per quanto riguarda il sistema *CADOCS* e il sistema *Slack*, non sarà possibile effettuare test di integrazione a causa della necessità di token e autenticazioni proprietarie di Slack, le quali non possono essere completamente simulate. Per questo motivo, considerando le risorse disponibili, queste interazioni saranno testate in modo globale nella fase di system testing.

Anche il test comprendente l'interazione con i sottosistemi csDetector e CADOCS_NLU sarà effettuato in fase di test di sistema a causa dell'eccessiva complessità nel simulare tali interazioni.

L'architettura di CADOCS (figura 4.1) sarà modificata con l'implementazione delle change requests, in particolare con l'implementazione della CR_4, ovvero il porting del sistema sulla piattaforma Discord.

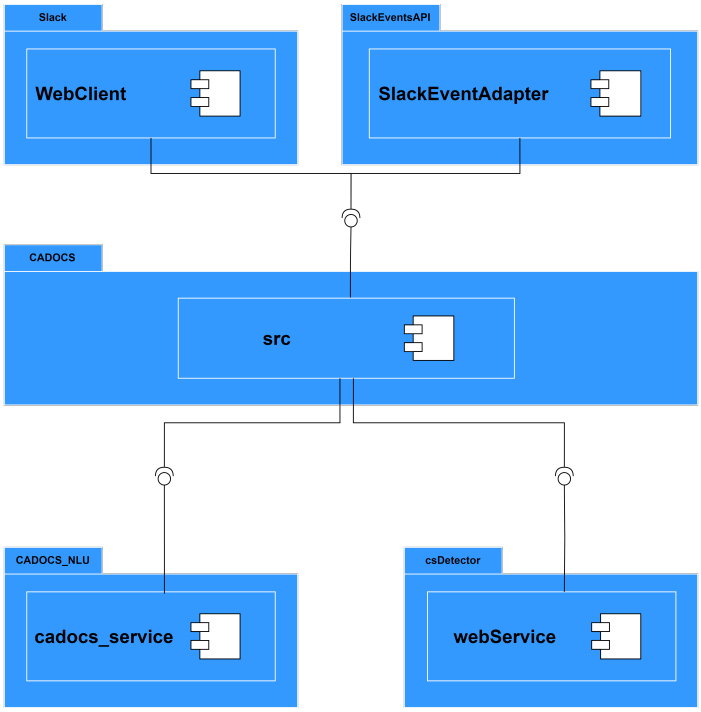

Figura 4.1: Architettura del sistema attuale

4.3 Testing di Sistema

Per la fase di testing di sistema, l'obiettivo è garantire l'integrità del comportamento del sistema implementato rispetto alle aspettative dell'utente. In questa fase sarà utilizzata una metodologia **black-box**. Per garantire la completezza della test suite, utilizzeremo la strategia *Category Partition*. Utilizzeremo Selenium IDE, un tool che permette di testare l'intero sistema partendo dall'interfaccia utente.

Il testing di sistema verrà implementato su quelle che sono state le funzionalità elencate nel capitolo 2, quindi si andrà a simulare l'inserimento di richieste da parte degli utenti sulla piattaforma Slack e successivamente, dopo modifica di porting di CADOCS su Discord, sulla piattaforma Discord. Una panoramica sull'architettura complessiva del sistema è riportata in Figura 4.1.

4.4 Testing di Regressione

L'approccio di testing di regressione sarà basato sui test di unità e di integrazione che saranno sviluppati sul sistema originale. In questo modo, si garantirà che le modifiche apportate al sistema o alle sue componenti non introdurranno nuovi bug o errori. Di conseguenza, ad ogni push nel branch *dev* tramite una GitHub Action, verranno eseguiti i test di unità e di integrazione per assicurare che il sistema continui a funzionare correttamente dopo ogni modifica.

4.5 Ispezione del Codice

Per valutare i contributi sul codice verranno effettuate delle ispezioni del codice sfruttando la funzionalità di "Code Review" fornita da Github, in modo da poter approvare o rigettare dei cambiamenti e poterne richiedere degli altri.

Capitolo 5

Test Case

Per affrontare il fattore di non determinismo del modello di NLU, sarà adottato un approccio basato su Input Testing, utilizzando una selezione rappresentativa di richieste direttamente tratte dal dataset utilizzato per l'addestramento.

Nel processo di definizione dei test case per la test suite, sarà inclusa la categoria "Intent".

Per i test case, sceglieremo diverse richieste tratte direttamente dal set di dati dell'input testing del modello utilizzato prima dell'implementazione delle modifiche, le quali sono già state verificate per essere classificate dal modello con un particolare intent. Questo ci consentirà di verificare con certezza che il modello identifichi correttamente gli intent.

Allo stesso tempo, per testare il caso di errore relativo agli intent, utilizzeremo richieste del tipo: "*aaaaaaaaaaaaa*", progettate appositamente per provocare un'evidente mancanza di intent o una classificazione erronea da parte del modello. Questi casi di test ci permetteranno di verificare la capacità del sistema di gestire richieste prive di significato. Occorre però considerare che utilizzando modelli di Natural Language Understanding, non è possibile prevedere con certezza uno specifico comportamento; questo implica che il successo di queste tipologie di test (con input poco significativi) potrebbero non essere garantito.

Inoltre, i test case forniti rappresentano una serie di scenari generici, ognuno dei quali corrisponde a un requisito specifico da verificare. Saranno poi implementati test specifici per testare sia la piattaforma Slack sia la piattaforma Discord, prendendo in

considerazione sia richieste in italiano che richieste in inglese, al fine di garantire una copertura completa delle funzionalità del nostro chatbot su entrambe le piattaforme e in entrambe le lingue supportate.

Tuttavia, è importante notare che per i test di sistema in pre-maintenance, saranno eseguiti esclusivamente sulla piattaforma Slack utilizzando esclusivamente richieste in lingua inglese.

Attraverso questo approccio di testing, miriamo a garantire che il nostro chatbot si comporti in modo affidabile e coerente su entrambe le piattaforme e che sia in grado di gestire correttamente richieste in diverse lingue. L'obiettivo finale è fornire un'esperienza utente ottimale e coerente indipendentemente dalla piattaforma utilizzata e dalla lingua in cui gli utenti interagiscono con il chatbot.

5.1 Test Case per RF get_smells

Descrizione	
Il sistema deve interpretare correttamente una richiesta di tipo get_smells_date	
Parametro: Richiesta	
Nome Categoria	Scelte
Intent [IR]	1. Tipo intent != get_smells [ERROR] 2. Tipo intent = get_smells [PROPERTY IR_OK]
Parametro: Link Formato: (?i)\b((?:https?://www\d{0,3}[.][a-z0-9.\-]+[.][a-z]{2,4}/)(?:[^\s()]+\(([^\s()]+(\([^\s()]+\)))*\))+(?:\(([^\s()]+(\([^\s()]+\)))*\)[^\s‘!()\[\]{};:’\".,?«»“”‘’]))	
Nome Categoria	Scelte
Formato [FL]	1. Formato link = false [ERROR] 2. Formato link = true [IF IR_OK] [PROPERTY FL_OK]

5.2 Test Case per RF get_smells_date

Descrizione	
Il sistema deve interpretare correttamente una richiesta di tipo get_smells_date	
Parametro: Richiesta	
Nome Categoria	Scelte
Intent [IR]	1. Tipo intent != get_smells_date [ERROR] 2. Tipo intent = get_smells_date [PROPERTY IR_OK]
Parametro: Link Formato: (?i)\b((?:https?://www\d{0,3}[.][a-z0-9.\-]+[.][a-z]{2,4}/)(?:[^\s()]+\(([^\s()]+(\([^\s()]+\)))*\))+(?:\(([^\s()]+(\([^\s()]+\)))*\)[^\s‘!()\[\]{};:’\".,?«»“”‘’]))	
Nome Categoria	Scelte
Formato [FL]	1. Formato link = false [ERROR] 2. Formato link = true [IF IR_OK] [PROPERTY FL_OK]
Parametro: Data Formato: MM/DD/YYYY	
Nome Categoria	Scelte
Formato [FD]	1. Formato data = false [ERROR] 2. Formato data = true [IF FL_OK] [PROPERTY FD_OK]
Validità [VD]	1. Validità data = false [ERROR] 2. Validità data = true [IF FD_OK] [PROPERTY VD_OK]

5.3 Test Case per RF report

Descrizione	
Il sistema deve interpretare correttamente una richiesta di tipo report	
Parametro: Richiesta	
Nome Categoria	Scelte
Intent [IR]	1. Tipo intent != report [ERROR] 2. Tipo intent = report [PROPERTY IR_OK]

5.4 Test Case per RF info

Descrizione	
Il sistema deve interpretare correttamente una richiesta di tipo info	
Parametro: Richiesta	
Nome Categoria	Scelte
Intent [IR]	1. Tipo intent != info [ERROR] 2. Tipo intent = info [PROPERTY IR_OK]

5.5 Test Suite

Test Case ID	Test Frame	Risultato
TC_GS_1	IR1	Errore: la richiesta non è interpretata correttamente
TC_GS_2	IR2, FL1	Errore: il formato del link non è corretto
TC_GS_3	IR2, FL2	Corretto

Test Case ID	Test Frame	Risultato
TC_GSD_1	IR1	Errore: la richiesta non è interpretata correttamente
TC_GSD_2	IR2, FL1	Errore: il formato del link non è corretto
TC_GSD_3	IR2, FL2, FD1	Errore: il formato della data non è corretto
TC_GSD_4	IR2, FL2, FD2, VD1	Errore: la data non è valida
TC_GSD_5	IR2, FL2, FD2, VD2	Corretto

Test Case ID	Test Frame	Risultato
TC_RP_1	IR1	Errore: la richiesta non è interpretata correttamente
TC_RP_2	IR2	Corretto

Test Case ID	Test Frame	Risultato
TC_IN_1	IN1	Errore: la richiesta non è interpretata correttamente
TC_IN_2	IN2	Corretto