LAUREA MAGISTRALE IN SOFTWARE ENGINEERING AND IT MANAGEMENT
UNIVERSITÀ DI SALERNO
CORSO DI INGEGNERIA GESTIONE EVOLUZIONE DEL SOFTWARE

PROF. ANDREA DE LUCIA
DOTT. MANUEL DE STEFANO,
EMANUELE IANNONE

Impact Analysis

Repository GitHub:

https://github.com/rebeccadimatteo/csDetector/tree/Explainability/explainability

https://github.com/rebeccadimatteo/CADOCS

2023

Rebecca Di Matteo

Leonardo Monaco

Indice

CAPITOLO 1

Contesto del Progetto

L'ingegneria del software è un'attività incentrata sull'uomo che coinvolge vari stakeholder con background diversi che devono comunicare e collaborare per raggiungere obiettivi condivisi. L'emergere di conflitti tra le parti interessate può portare a effetti indesiderati sulla manutenibilità del software, ma è spesso inevitabile nel lungo periodo.
I Community Smells, ovvero le pratiche di comunicazione e collaborazione non ottimali, sono stati definiti per individuare i conflitti ricorrenti tra gli sviluppatori.
Per facilitare un uso più ampio delle informazioni relative ai Community Smells da parte dei praticanti, prendiamo in considerazione CADOCS, un agente conversazionale client-server che si basa su un precedente strumento di rilevamento di Community Smells: CSDETECTOR. Quest'ultimo è uno strumento che rileva automaticamente i Community Smells con un approccio basato sull'apprendimento automatico; esso parte dall'analisi dei repository pubblici su GITHUB ed utilizza l'elaborazione del linguaggio naturale per comprendere l'intento dietro i messaggi di commit degli sviluppatori, le richieste pull, e problemi. Con questa conoscenza, lo strumento calcola le metriche socio-tecniche, che verranno utilizzate per addestrare un modello di apprendimento automatico in grado di prevedere i Community Smells su determinati repository.
L'idea alla base di questo progetto è di potenziare CADOCS aggiungendo nuovi moduli basati su ML, andando a creare dei modelli di Explainability all'interno di CSDETECTOR così da comprendere la spiegabilità del tool nel determinare le metriche socio-tecniche e la presenza o non dei Community Smells.

CAPITOLO 2

Obiettivi del progetto

Per fornire agli utenti nuove conoscenze, si vuole raffinare il tool CADOCS in modo da dare ai praticanti uno strumento migliore in grado di esprimere anche l'Explainability dei modelli che ricavano, mediante le metriche socio-tecniche, la presenza dei Community Smells. Explainable Artificial Intelligence è un insieme di metodi e processi che consentono agli utenti di comprendere e considerare attendibili i risultati e l'output creati dagli algoritmi di Machine Learning. L'intelligenza artificiale spiegabile viene utilizzata per descrivere un modello di intelligenza artificiale, il relativo impatto previsto ed i potenziali errori.
Le funzionalità del tool verranno estese mediante l'analisi della spiegabilità dei modelli di predizione dei Community Smells per mezzo di due librerie: **Python LIME e SHAP**, al fine di comprendere quali metriche socio-tecniche siano più influenti nella predizione dei suddetti. L'Explainability sarà inserita all'interno di CADOCS mediante un' interfaccia grafica che rappresenterà l'output di una delle due librerie. La scelta dell'Explainability da rappresentare sarà affettuta sulla base di uno studio relativo ai risultati di un questionario che, sarà somministrato agli studenti del Dipartimento di Informatica dell'Università degli Studi di Salerno. Nel questionario verranno rappresentati gli output delle due librerie con delle domande a riguardo. I risultati ottenuti ci permetteranno di rispondere alle domande di ricerca che ci siamo posti per comprendere quale fra i due framework ha un'Explainability più chiara e una predizione più veritiera nel determinare le metriche socio-tecniche per la ricerca dei Community Smells, così da rappresentarla graficamente su CADOCS. Nello stato attuale CADOCS non contiene un approfondimento relativo all'Explainability. L'inserimento di tale

approfondimento può portare l'utente ad avere una maggiore fiducia sui risultati del processo decisionale che porta all'identificazione dei Community Smells. Inoltre, basandosi su questi risultati l'utente può comprendere quale metrica socio-tecnica è più influente rispetto ad un'altra nel determinare il Community Smells e dunque attuare delle strategie per risolverlo.

CAPITOLO 3

Stato dell'Arte : CADOCS & CSDETECTOR

CADOCS è un agente conversazionale che lavora sulla piattaforma Slack ed è in grado mediante CSDETECTOR di identificare e gestire i Community Smells nelle comunità di sviluppo software su GitHub. CSDETECTOR è uno strumento che rileva automaticamente i Community Smells con un approccio di apprendimento automatico. Partendo dall'analisi dei repository pubblici su GITHUB, utilizza l'elaborazione del linguaggio naturale per comprendere l'intento dei messaggi di commit, delle richieste di pull e dei problemi degli sviluppatori. Con questa conoscenza, lo strumento calcola le metriche socio-tecniche, che vengono utilizzate per addestrare un modello di apprendimento automatico in grado di predire i Community Smells all'interno di determinate repository. Per raggiungere il nostro obiettivo, abbiamo in programma di estendere le funzionalità di CADOCS.

Dal punto di vista dei requisiti svilupperemo:

- un potenziamento di CADOCS aggiungendo nuovi moduli basati su ML.
  Inserendo all'interno di CSDETECTOR un modello basato sull'Explainability che vada ad analizzare l'output restituito da quest'ultimo tool
- una rappresentazione grafica dell'Explainability su CADOCS
- un refactoring del codice per migliorare la chiarezza e l'organizzazione del codice.

CAPITOLO 4

Analisi del Sistema

Prima di raffinare il sistema abbiamo attuato un analisi di CSDETECTOR e CADOCS. Così da comprendere come i due tool fossero stati progettati ed implementati.

4.0.1 CSDETECTOR

In una prima fase abbiamo analizzato il Class Diagram sottostante, al fine di comprendere come fossero suddivisi i moduli ed i relativi metodi implementati, successivamente ci siamo soffermati sulla comprensione di ogni modulo. Abbiamo analizzato le dipendenze e i requisiti funzionali che hanno portato il progettista a creare lo strumento.

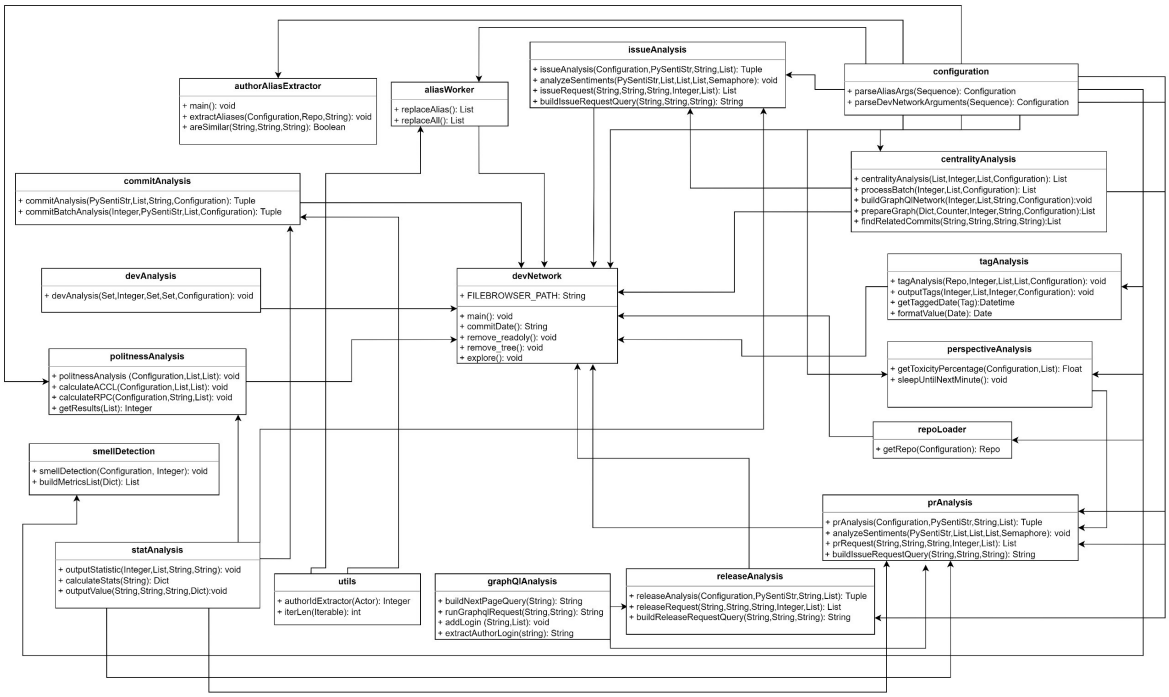

Figura 4.1: Class Diagram CSDETECTOR

4.0.2 Analisi delle dipendenze CSDETECTOR

Abbiamo analizzato la matrice delle dipendenze raffigurata nell'immagine sottostante. La quale mostra il legame fra l'utilizzo di una funzionalità di un modulo in un altro, inserendo nella cella in numero di volte in cui ciò avviene.

Osservando la matrice delle dipendenze possiamo notare che, ci sono due moduli che hanno forte dipendenze. In particolare :

- **devNetwork.py :**
  la sua riga riportata nella matrice delle dipendenze con id 6, con una maggioranza di valori non nulli, ciò implica che usa molte funzioni di altri moduli.

- **configuration.py :**
  la sua colonna riportata nella matrice delle dipendenze con id 4, con una maggioranza di valori non nulli, ciò implica che usa molte funzioni di altri moduli.

Modulo	id	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
aliasWorker	**0**	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0
authorAliasExtractor	**1**	0	0	0	0	2	0	0	0	0	1	0	0	0	2	0	0	0	0
centralityAnalysis	**2**	0	0	0	0	1	0	0	0	0	0	0	5	0	3	0	0	0	0
commitAnalysis	**3**	0	0	0	0	1	0	0	0	0	0	0	7	0	1	0	0	0	0
configuration	**4**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
devAnalysis	**5**	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
devNetwork	**6**	1	0	2	1	1	1	0	0	1	1	1	0	1	0	0	1	1	1
perspectiveAnalysis	**7**	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
politenessAnalysis	**8**	0	0	0	0	1	0	0	0	0	0	0	2	0	0	0	0	0	0
repoLoader	**9**	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
smellDetection	**10**	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
statsAnalysis	**11**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
tagAnalysis	**12**	0	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
utils	**13**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
graphqlAnalysisHelper	**14**	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
issueAnalysis	**15**	0	0	1	0	1	0	0	1	0	0	0	7	0	0	3	0	0	0
prAnalysis	**16**	0	0	1	0	1	0	0	1	0	0	0	8	0	0	3	0	0	0
releaseAnalysis	**17**	0	0	0	0	2	0	0	0	0	0	0	2	0	0	2	0	0	0

Dalle analisi precedenti possiamo dedurre che :

- l'intera esecuzione dello strumento è gestita dal modulo **devNetwork.py**
- **configuration.py** è responsabile di mantenere lo stato del programma durante la sua esecuzione (infatti è l'unico modulo implementato come una vera e propria CLASSE).
- CSDETECTOR viene eseguito da **devNetwork** proprio come un programma procedurale, utilizzando le informazioni memorizzate in un Configuration object.

4.0.3 Analisi Requisiti CSDETECTOR

Abbiamo analizzato i requisiti dello strumento originale. Lo scopo del sistema è il rilevamento dei Community Smells, in tale processo ci sono molti passaggi coinvolti, che possono essere considerati come requisiti.

CSDETECTOR Requisiti Funzionali		
RF_1	Rilevamento dei Community Smells	Il sistema deve essere in grado di rilevare e mostrare i Community Smells su un determinato repository GitHub.
RF_2	Gestione del repository GitHub	Il sistema deve essere in grado di scaricare un determinato repository GitHub e di accedere al suo contenuto.
RF_3	Calcolo delle metriche del progetto	Il sistema deve essere in grado di raccogliere
RF_4	Persistenza delle metriche	Il sistema deve essere in grado di scrivere e leggere localmente le metriche del progetto in un account.

Tabella 4.1: Requisiti Funzionali CSDETECTOR

4.0.4 Sequence Diagram CSDETECTOR

Per comprendere meglio il Tool, abbiamo analizzato le attività dello strumento nelle quattro fasi logiche, mostrate nella figura sottostante.

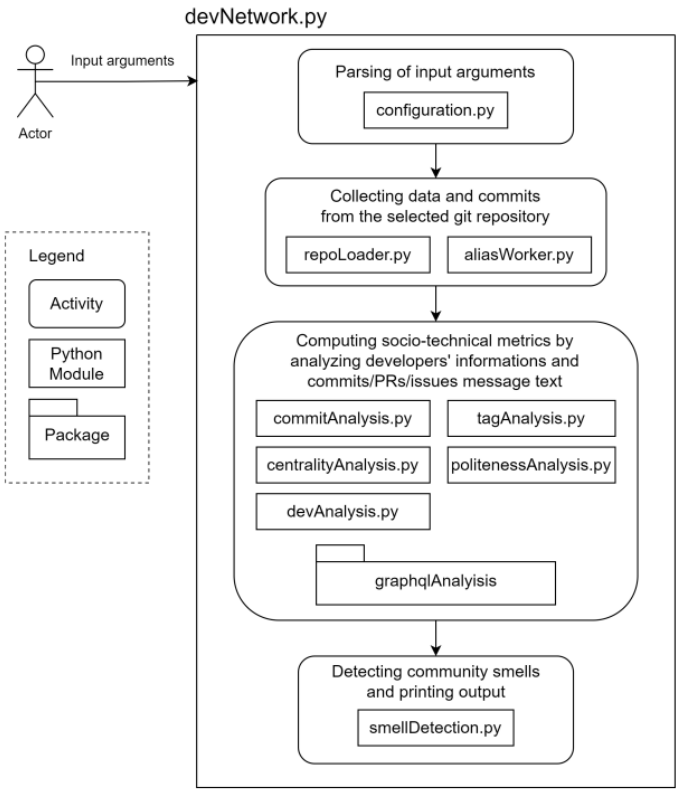

Figura 4.2: Eseguzione CSDETECTOR gestito da devNetwork.py.

- **Macro-module devNetwork.py:**
  Può essere visto, come un wrapper per l'intera esecuzione dello strumento.

- **Parsing of input arguments:**
  In questa fase del processo, viene creato un oggetto Configuration e viene inizializzato con gli input passati come argomenti dall'utente. Può essere visto come una parte dell'RF_1, cioè il modo in cui lo strumento ottiene l'URL del repository dall'utente.

- **Collecting data and commits from the selected git repository:**
  In questa fase, vengono utilizzate le informazioni di configurazione del repository git e viene scaricato localmente, raccogliendo tutto il necessario. In questo processo vien soddisfatto il requisto RF_2.

- **Computing socio-technical metrics by analyzing developers' information and commitsPRsissues message text:**
  Questa fase coinvolge in realtà due requisiti funzionali - RF_3 e RF_4 ed è il modulo centrale che raccoglie i dati necessari per eseguire una previsione sui moduli pre-addestrati per rilevare i Community Smells.

- **Detecting community smells and printing output:**
  E' l'ultima fase dell'esecuzione, in cui il sistema esegue l'RF_1 E rileva i Community Smells sul repository dato.

4.1 CADOCS

In una prima fase abbiamo analizzato l'architettura di CADOCS rappresentata nell'immagine sottostante, successivamente abbiamo creato un Class Diagram, al fine di comprendere come fossero suddivisi i moduli ed i relativi metodi implementati, successivamente ci siamo soffermati sulla comprensione di ogni modulo. Abbiamo analizzato le dipendenze e i requisiti funzionali che hanno portato il progettista a creare lo strumento.

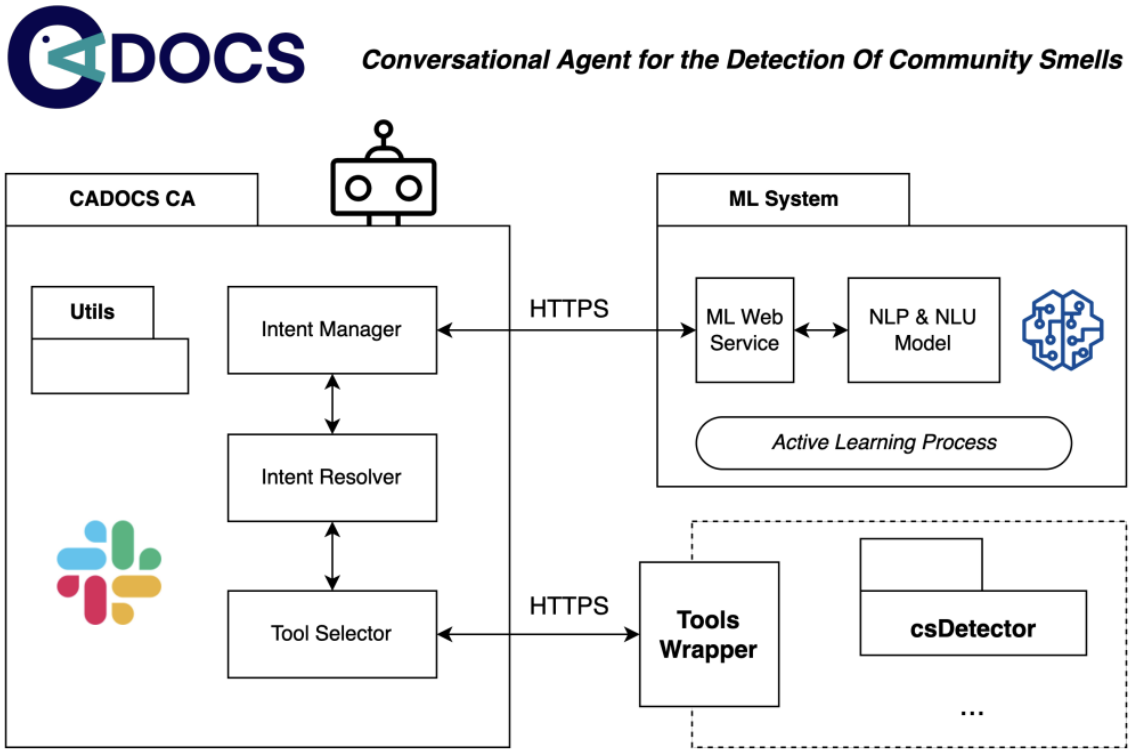

Figura 4.3: Architettura CADOCS

CADOCS è stato progettato per aderire a un'architettura client-server, l'architettura di riferimento per la costruzione di agenti conversazionali. Come illustrato nella Figura, lo strumento è suddiviso in tre moduli.

- **Tools Wrapper**

  Nella sua versione iniziale, CADOCS è stato progettato per avvolgere CSDETECTOR. Ciò significa che il nostro strumento rende attualmente disponibili le funzionalità fornite da CSDETECTOR, sia in termini di misurazione delle metriche socio-tecniche che di rilevamento dei Community Smells.

- **Machine Learning System**

  Il machine learner alla base dello strumento ha l'obiettivo di facilitare l'interazione tra gli utenti e la logica del sistema. Tale interazione è stata pensata in termini degli obiettivi che l'utente ha in mente quando interroga il bot, e sono stati implementati attraverso l'elaborazione del linguaggio naturale (NLP) e la comprensione del linguaggio naturale (NLU). Inoltre, il modulo ha un meccanismo di apprendimento attivo per garantire la crescente capacità del bot di riconoscere le intenzioni degli utenti e migliorare la sua usabilità nel corso del tempo.

- **Conversational Agent**

  È la parte centrale del bot, contiene la logica per interagire con l'utente. Interagisce con gli spazi di lavoro di Slack ed è stato implementato utilizzando API degli eventi di Slack. Il modulo contiene anche la logica per suggerire strategie di refactoring. Funziona con un end-point dell'API in grado di catturare gli eventi che si verificano nello spazio di lavoro Slack autenticato attraverso token personali. Ogni volta che CADOCS rileva un nuovo messaggio, scritto in uno qualsiasi dei canali canali in cui è stato aggiunto, si avvia l'intero processo. Dopo aver calcolato il messaggio giusto, in base all'intento rilevato dal modello ML, pubblica la risposta nel canale da cui il messaggio è stato ricevuto.

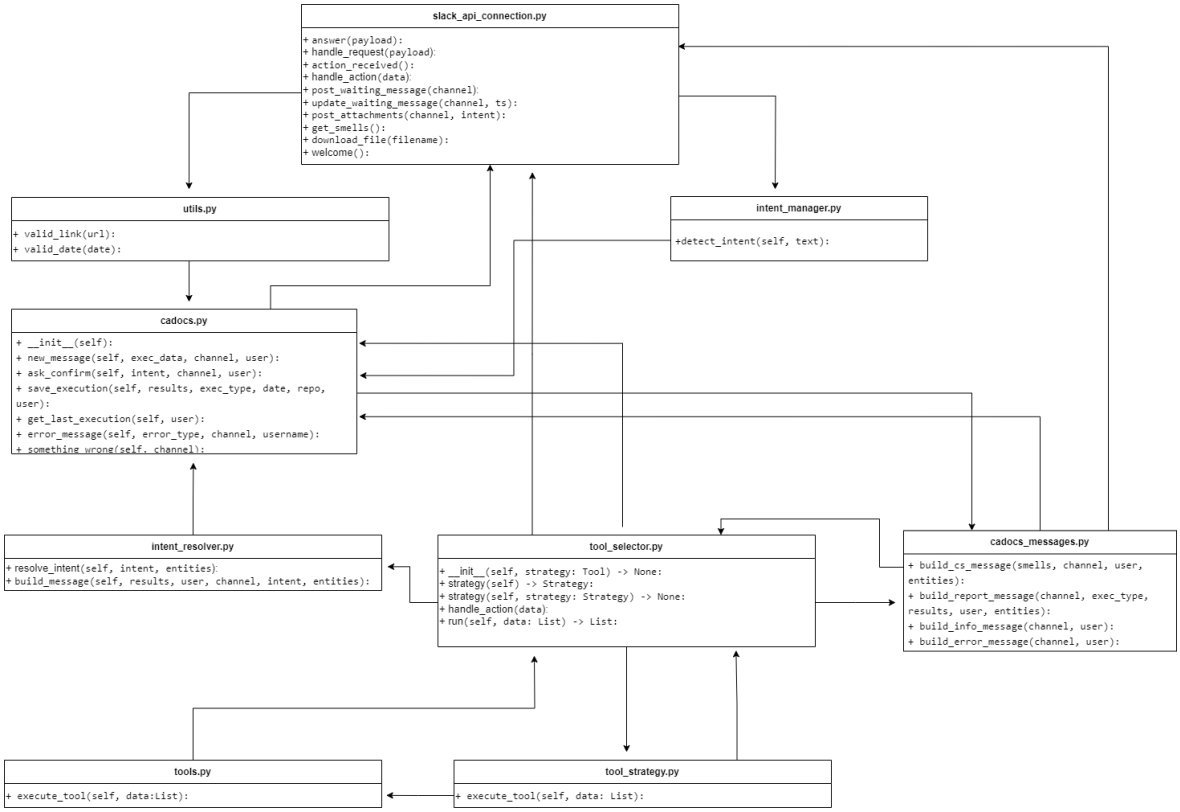

Figura 4.4: Class Diagram CADOCS

Dal Class Diagram creato analizzando CADOCS, abbiamo compreso la suddivisione dei moduli ed i relativi metodi implementati.

4.1.1 Analisi Dipendenze CADOCS

Abbiamo analizzato la matrice delle dipendenze raffigurata nell'immagine sottostante. La quale mostra il legame fra l'utilizzo di una funzionalità di un modulo in un altro, inserendo nella cella in numero di volte in cui ciò avviene. Osservando la matrice delle dipendenze possiamo notare che, non ci sono forti dipendenze tra i moduli oltre quella del modulo **tool_strategy**, i quali metodi sono richiami in diversi modduli.

Modulo	ID	0	1	2	3	4	5	6	7	8
cadocs_messages.py	0	0	1	0	0	2	1	0	0	0
cadocs.py	1	1	0	0	0	3	0	0	0	0
intent_resolver.py	2	0	2	0	0	0	0	0	0	0
intent_manager.py	3	0	1	0	0	0	0	0	0	0
slack_api_connection.py	4	0	0	0	1	0	0	0	0	1
tool_selector.py	5	1	2	1	0	2	0	1	0	0
tool_strategy.py	6	0	0	0	0	0	1	0	1	0
tools.py	7	0	0	0	0	0	1	0	0	0
utils.py	8	0	2	0	0	0	0	0	0	0

Tabella 4.2: Matrice delle dipendenze CADOCS

4.1.2 Analisi Requisiti CADOCS

Abbiamo analizzato i requisiti del sistema e abbiamo aggiunto un nuovo requisito **RF_6** in quanto andremo a rappresentare su CADOCS oltre che i Community Smells, l'Explainability con cui il modello di ML riesce mediante le metriche socio-tecniche a predire il Community Smells.

CADOCS Requisiti Funzionali		
RF_1	Rilevare Community Smells	Il sisteme deve riuscire a rilevare i Communty Smells ed altre informazioni socio-tecniche correlate, fornite da CSDETECTOR
RF_2	Rilevare Community Smells per data	Il sisteme deve riuscire a rilevare i Communty Smells ed altre informazioni socio-tecniche da una determinata data in poi
RF_3	Mostrare un report dell'ultima esecuzione	Il sistema deve mostrare un report dell'ultima esecuzione su Slack
RF_4	Rappresentazione Community Smells	Il sistema deve riuscire a rappresentare informazioni sui Community Smells
RF_5	Suggerire strategie di Refactoring	Il sistema deve suggerire strategie di Refactoring per i Community Smells rilevati
RF_6	Rappresentare Explainability	Il sistema deve riuscire a rappresentare Explainability di come il modello di ML riesce ad identificare i Community Smells nelle repository GitHub

Tabella 4.3: Requisiti Funzionali CADOCS

CAPITOLO 5

Proposed Change Request

In questa sezione, si spiegherà come si intende modificare e migliorare lo strumento CSDETECTOR e CADOCS. Le change request saranno elencate ed eseguite in ordine di priorità.
Abbiamo tre gradi di priorità :

- alto
- medio
- basso

Le change request con priorità alta saranno sviluppate per prime poi successivamente quelle a priorità media e infine quelle a priorità bassa, il grado di priorità va a definire la necessità di implementare alcune modifiche prima di altre. Quando andiamo ad effettuare delle modifiche dobbiamo considerare l'effort di quest'ultime.
Abbiamo tre gradi di effort :

- alto
- medio
- basso

Le change request con un effort alto hanno una quantità di lavoro maggiore rispetto ad altre con effort medio o basso.

5.1 CR_1 Estensione di CSDETECTOR con modulo di ML basato sull'Explainability

Descrizione
Creazione del modello basato sull'Explainability che prende in input le metriche socio-tecniche e i Community Smells forniti in output da CSDETECTOR. Il modello sarà eseguito su due librerie Python LIME e SHAP che forniranno in output il grado di Explainability.
Motivazione
Nello stato attuale CADOCS non contiene un approfondimento relativo all'Explainability. L'inserimento di tale approfondimento può portare l'utente ad avere una maggiore fiducia sui risultati del processo decisionale che porta all'identificazione dei Community Smells, inoltre basandosi su questi risultati quest'ultimo può comprendere quale metrica socio-tecnica è più influente rispetto ad un altra nel determinare il Community Smells e dunque attuare delle strategie per risolverlo.
Priority
Alta [X] Media [] Bassa[]
Effort
Alta [X] Media [] Bassa[]
Conseguenze della non accettazione
Il tool non contenendo un approfondimento sull'Explainability, non faciliterà l'utente nella comprensione dei Community Smells estratti e nella risoluzione di quest'ultimi.

5.1.1 Metodologia

L'obiettivo di questa **CR** è quello di costruire un modello di predizione per ogni Community Smells, una volta costruiti eseguirli su due librerie Explanable. Per svolgere questo dovremmo:

- Nella prima fase analizzare le diverse librerie Explainable focalizzandoci sulle caratteristiche positive e negative, per poi concentrarci su due librerie nello sepcifico.
- Nella seconda fase costruire un modello di predizione per ogni Community Smells. Tali modelli saranno successivamente eseguiti sulle due librerie scelte nella prima fase.

- Nella terza fase analizzare l'output fornito dalle due librerie e sulla base di esso sviluppare due questionari. Il primo per comprendere quale fra le due abbia una spiegabilità migliore, per poi rappresentare la più spiegabile su CADOCS. Il secondo per comprendere quanto la predizione dei due framework è conforme alla realtà, così da testare il grado di verità di ciò che verrà rappresentato su CADOCS.
- Nella quarta fase analizzare i risultati dei questionari, e sulla base di essi decidere quale output fra le due librerie rappresentare graficamente su CADOCS.

5.1.2 Risultati attesi

Se questa modifica verrà accettata, apporterà un miglioramento a CADOCS, in quanto gli utenti che lo utilizzeranno potranno comprendere quale metrica-socio tecnica è più rilevante per la predizione del Community Smells ed attuare delle strategie per risolverlo.

5.2 CR_2 Estensione di CADOCS con un interfaccia grafica per rappresentare l'Explainability

Descrizione
Creazione di un interfaccia grafica per rappresentare l'Explainability all'interno di CADOCS.
Motivazione
Nello stato attuale CADOCS non contiene una rappresentazione grafica dell'Explainability. Tale inserimento può portare facilità e immediatezza per l'utente nel comprendere come si arrivi ad avere determinati Community Smells e sfruttare tali risultati per applicare delle strategie per risolverli.
Priority
Alta [X] Media [] Bassa[]
Effort
Alta [X] Media [] Bassa[]
Conseguenze della non accettazione
Il tool non contenendo un interfaccia grafica rappresentante l'Explainability non faciliterà l'utente nella comprensione dei Community Smells estratti.

5.2.1 Metodologia

L'obiettivo di questa **CR** è quello di rappresentare l'Explainability su CADOCS, così che gli utenti utilizzatori del Tool possano comprendere, quale metrica è più influente nella predizione del Community Smells. Per svolgere questo compito andremo a modificare il messaggio dato in output da CADOCS su Slack inserendo al suo interno un messaggio con :

- il nome della libreria dalla quale deriva l'analisi
- il Community Smells
- la metrica socio-tecnica più influente per il Community Smells
- un suggerimento su come risolvere il Community Smells

5.2.2 Risultati attesi

Se questa modifica verrà accettata, apporterà un miglioramento a CADOCS, in quanto faciliterà l'utente nella comprensione dei Community Smells estratti e gli fornirà dei suggerimenti per risolverli.

5.3 CR_3 Refactoring del codice

Descrizione
Riorganizzare, ristrutturare e rendere più chiaro il codice esistente di CADOCS garantendo che il comportamento complessivo del codice non cambi.
Motivazione
Il Refactoring del codice potrebbe rendere CADOCS maggiormente organizzato, più chiaro e leggibile.
Priority
Alta [] Media [X] Bassa[]
Effort
Alta [] Media [X] Bassa[]
Conseguenze della non accettazione
Non attuando il Refactoring del codice il tool potrebbe risultare poco chiaro e poco organizzato.

5.3.1 Metodologia

L'obiettivo di questa **CR** è quello di riorganizzare, ristrutturare e rendere più chiaro il codice esistente di CADOCS garantendo che il comportamento complessivo del codice non cambi. Per svolgere questo compito andremo ad :

- Identificare cosa rifattorizzare;
- Determinare quale refactoring applicare
- Assicurarsi che il refactoring preserva il comportamento del software;
- Applicare la rifattorizzazione alle entità scelte;
- Valutare l'impatto della rifattorizzazione;
- Mantenere la consistenza.

5.3.2 Risultati attesi

Se questa modifica verrà accettata, apporterà un miglioramento a CADOCS, rendendolo maggiormente organizzato, più chiaro e leggibile.

CAPITOLO 6

Impact Analysis

Poiché le nostre modifiche potrebbero avere un effetto collaterale sul resto del sistema, abbiamo analizzato l'impatto di ogni modifica identificando l'insieme dei moduli che possono essere interessati. Abbiamo iniziato identificando quale requisito funzionale fosse coinvolto nella modifica, in modo da poter effettuare una mappatura tra la funzionalità e il modulo utilizzato. Inoltre, abbiamo raccolto informazioni aggiuntive utilizzando:

- la matrice delle dipendenze
- il codice sorgente

Per ogni CR, abbiamo creato una tabella con i moduli ed i possibili impatti all'interno di CADOCS, possiamo avere cinque tipi di impatti:

- **Starting impacy set**
  L'insieme iniziale di oggetti (o componenti) che presumibilmente verranno impattati dalla CR.
- **Cadidate Impact Set**
  L'insieme di oggetti (o componenti) che si stima possano essere impattati secondo un certo approccio di impact analysis
- **Discovered Impact Set**
  L'insieme di nuovi oggetti (o componenti), non contenuti nel CIS, che si scopre essere influenzato durante l'implementazione di una CR.

- **Actual Impact Set**
  L'insieme di oggetti (o componenti) che è effettivamente cambiato in seguito all'esecuzione di una CR.

- **False Positive Impact Set**
  L'insieme di oggetti (o componenti) che si stima siano influenzati da un'implementazione di una CR ma che non sono effettivamente influenzati dalla CR.

Con queste conoscenze, saremo in grado di raffinare il tool senza preoccuparci di introdurre difetti.

6.0.1 CR_1 Estensione di CSDETECTOR con modulo di ML basato sull'Explainability

Per quando riguarda la **CR_1**, ovvero la creazione dei modelli di ML per la predizione dei Community Smells e la successiva eseguzione di tali modelli sulle librerie Explainable non impatta sui due sistemi, in quanto saranno creati esternamente ad essi.

IMPACT SETS FOR CR_1					
Modulo	**Starting Impact Set**	**Candidate Impact Set**	**Discovered Impact Set**	**Actual Impact Set**	**False Positive Impact Set**
cadocs_messages.py					
cadocs.py					
intent_resolver.py					
intent_manager.py					
slack_api_connection.py					
tool_selector.py					
tool_strategy.py					
tools.py					
utilis.py					

Tabella 6.1: Impact Sets for CR_1

6.0.2 CR_2 Estensione di CADOCS con un interfaccia grafica per rappresentare l'Explainability

Per quando riguarda la **CR_2**, ovvero l'estensione di CADOCS mediante un interfaccia grafica per rappresentare l'Explainability, andremo a modificare il messaggio che darà in output CADOCS su Slack aggiugendo le informazioni riguardanti l'Explainability all'interno del file **community_smells.json**. Analizzando le dipendenze e il codice possiamo notare che essa può avere un impatto all'interno di diversi moduli:

- **cadocs.py e cadocs_messages.py** in quanto richiama il metodo sopra descritto
- **slack.api.connection.py** in quanto ha delle dipendenze con cadocs.py e cadocs_messages.py
- **tool_selector.py** in quanto ha delle dipendenze con cadocs_messages.py
- **intent_manager.py** in quanto ha delle dipendenze con slack.api.connection.py
- **utils.py** in quanto ha delle dipendenze con slack.api.connection.py
- **intent_resolver.py** in quanto ha delle dipendenze con **tool_selector.py** e **tool_strategy.py**

IMPACT SETS FOR CR_2					
Modulo	**Starting Impact Set**	**Candidate Impact Set**	**Discovered Impact Set**	**Actual Impact Set**	**False Positive Impact Set**
cadocs_messages.py	x	x		x	
cadocs.py	x	x		x	
intent_resolver.py			x		x
intent_manager.py			x		x
slack_api_connection.py			x		x
tool_selector.py			x		x
tool_strategy.py			x		x
tools.py					
utilis.py			x		x

Tabella 6.2: Impact Sets for CR_2

6.0.3 CR_3 Refactoring del codice

Per quando riguarda la **CR_3**, ovvero la riorganizzazione e ristrutturazione del codice esistente di CADOCS. Analizzando le dipendenze e il codice possiamo notare che essa può avere un impatto all'interno di tutti i moduli, in quanto andremo a riorganizzare e ristrutturare tutti i moduli cercando di rendere il codice più chiaro e leggibile.

IMPACT SETS FOR CR_3					
Modulo	**Starting Impact Set**	**Candidate Impact Set**	**Discovered Impact Set**	**Actual Impact Set**	**False Positive Impact Set**
cadocs_messages.py	x	x		x	
cadocs.py	x	x		x	
intent_resolver.py	x	x		x	
intent_manager.py	x	x		x	
slack_api_connection.py	x	x		x	
tool_selector.py	x	x		x	
tool_strategy.py	x	x		x	
tools.py	x	x		x	
utilis.py	x	x		x	

Tabella 6.3: Impact Sets CR_3

CAPITOLO 7

Attuazione delle modifiche

Dopo aver analizzato i possibili impatti delle modifiche sul sistema, abbiamo effettuato il fork delle repository originali su GITHUB per apportare le nostre modifiche.
Repository GitHub:
https://github.com/rebeccadimatteo/csDetector/tree/Explainability/explainability
https://github.com/rebeccadimatteo/CADOCS
In questa sezione forniremo una rapida panoramica delle tecnologie utilizzate nel progetto per attuare le modifiche e del loro funzionamento.

7.1 CR_1 Estensione di CSDETECTOR con modulo di ML basato sull'Explainability

Per estendere CSDETECTOR inserendo un modulo di ML basato sull'Explainability, andremo a creare un modello di predizione per ogni Community Smells, che prende in input le metriche socio-tecniche e predice il Community Smells. Abbiamo utilizzato la piattaforma **Google Colab** e il linguaggio di programmazione Python per costruire i modelli, successivamente li abbiamo eseguiti su due librerie Explanable **LIME e SHAP**, dopo aver analizzato diversi framework di Explanable, si è deciso di utilizzare i framework LIME e SHAP, poichè risultano essere interessanti in quanto tra di loro rappresentano due tipi di spiegabilità differenti in diversi fattori, ciò ci permette di fornire una spiegabilità completa sotto diverse caratteristiche.

Nello specifico LIME è una tecnica post-hoc ovvero, fornisce spiegazioni locali su ogni singola osservazione di un dataset interpretabile da un qualsiasi classificatore di machine learning, andando ad analizzare le caratteristiche di un input e le previsioni.
SHAP è un metodo per spiegare le previsioni individuali, basato sui valori Shapley, teoricamente ottimali del gioco. Il valore di Shapley è il contributo marginale medio di un valore di una caratteristica in tutte le possibili coalizioni con le altre caratteristiche, ad esempio se prendessimo in considerazione un gioco con dei giocatori ed una vincita, il valore di Shapley rappresenta il contributo di quel giocatore nel gioco al fine di arrivare alla vittoria. Sia LIME che SHAP risultano esprimere una spiegazione a livello locale, ma con SHAP possiamo ottenere anche una spiegazione a livello globale fornendo informazioni sull'insieme delle osservazioni di un dataset interpretabile, andando ad aggregare tutte le spiegazioni locali.
L'output di LIME può essere presentato graficamente con il metodo "show in notebook". Quest'ultimo mostra nella parte destra le diverse features e il valore assegnato ad esse, nella parte sinistra la probablità della predizione della variabile dipendente. SHAP rappresenta in output la feature più promettente al fine della predizione; per rappresentare ciò vi è la possibilità di utilizzare diversi tipi di grafici.
Successivamente è stato analizzato l'output fornito dalle due librerie e sulla base di esso sono stati sviluppati due questionari.
Il primo per comprendere quale fra LIME e SHAP avesse una spiegabilità migliore, per poi rappresentare il più spiegabile su CADOCS.
Il secondo per comprendere quanto la predizione dei due framework fosse conforme alla realtà, così da testare il grado di verità di ciò che verrà rappresentato su CADOCS.
Per sviluppare i questionari abbiamo utilizzato la piattaforma **Google form**. Infine abbiamo analizzato i risultati dei questionari, e sulla base di essi è stato deciso di rappresentare graficamente SHAP su CADOCS, in quanto risulta essere più spiegabile rispetto a LIME. Inoltre le sue predizioni risultano essere conformi alla realtà.

7.2 CR_2 Estensione di CADOCS con un interfaccia grafica per rappresentare l'Explainability

Per estendere CADOCS con un interfaccia grafica per rappresentare Explainability siamo andati a modificare il file **Community_smells.json** inserendo all'interno del messaggio che da in output CADOCS sulla piattaforma Slack :

- il nome della libreria dalla quale deriva l'analisi
- il Community Smells
- la metrica socio-tecnica più influente per il Community Smells
- un suggerimento su come risolvere il Community Smells

7.3 CR_3 Refactoring del codice

Per attuare il refactoring del codice, siamo andati ad analizzare tutti i moduli presenti all'interno di CADOCS. In una prima fase abbiamo identificato cosa rifattorizzare, analizzando il codice sorgente e soffermandoci sulle classi,funzioni, metodi e dati.
In una seconda fase abbiamo applicato il refcatoring.

La prima strategia di rafctoring apllicata consiste nel rinominare alcuni metodi e variabili come:

1. Il modulo **utils.py** rinominando:
    - Il nome del metodo **valid_link** in **is_valid_link**, per rendere più chiaro il suo scopo, dato che questo metodo era presente anche nel moldulo CADOCS è stato modoficato anche in quest'ultimo.
    - Il nome del metodo **valid_date** in **is_valid_date**, per rendere più chiaro il suo scopo, dato che questo metodo era presente anche nel moldulo CADOCS è stato modoficato anche in quest'ultimo.
    - Il nome della variabile **re_equ** è stato cambiato in **regex_eq** per indicare che contiene un'espressione regolare.
    - Il nome della variabile **rs_date** è stato cambiato in **result_date** per indicare che contiene il risultato di una ricerca.
    - Il nome della variabile **ls** è stato cambiato in **date_parts** per indicare che rappresenta le parti di una data.
2. Il modulo **intent_manager.py** rinominando:

- La variabile **req** in **nlu_request** per renderne più chiara la sua utilità.

3. Il modulo **cadocs.py** rinominando:

   - Il metodo **ask_confirm** in **build_confirmation_message** per renderne più chiara la sua utilità.
   - Il metodo **something_wrong** in **build_error_response** dato che questo metodo era presente anche nel moldulo sack.apy.connection è stato modoficato anche in quest'ultimo.

   - La variabile **exec_data** in **execution_date**.

4. Il modulo **cadocs_messange.py** rinominando:

   - Il metodo **build_cs_message** in **build_community_smells_message**.

Le caratteristiche su cui ci siamo basati per effettuare il refactoring sono :

- **Chiarezza**: I nuovi nomi sono stati scelti in modo da comunicare in modo più chiaro lo scopo e la funzionalità dei metodi e delle variabili. Questo rende più facile per gli sviluppatori capire cosa fa ogni parte del codice senza dover esaminare in dettaglio l'implementazione.
- **Leggibilità:** I nuovi nomi sono più leggibili e comprensibili rispetto a quelli originali. Sono stati scelti in modo da utilizzare parole significative e evitare abbreviazioni o acronimi poco chiari.
- **Convenzioni di denominazione:** Abbiamo adottato le convenzioni di denominazione di Python, come l'uso di lettere minuscole e separazione delle parole con underscore per i metodi e le variabili. Questo aiuta a rendere il codice più uniforme e coerente con gli standard di codifica Python.
- **Coerenza:** Abbiamo cercato di mantenere una coerenza nella denominazione dei metodi e delle variabili all'interno del codice, in modo che siano facili da riconoscere e ricordare.

Oltre a rifattorizzare il codice, rinominado metodi e variabili, abbiamo analizzato il codice sorgente per identificare se ci fossero duplicazioni di codice così da estrarle e creare un metodo che potesse essere richiamato in diverse parti del codice. Non abbiamo identificato nessuna duplicazione all'interno del codice che potesse essere estratta per migliorare la leggibilità di quest'ultimo.

Inoltre abbiamo reso il codice più leggibile per chi lo consulta, inserendo degli spazi tra i diversi metodi nei moduli.

Nella terza fase ci siamo assicurati che il refactoring preservasse il comportamento del software mediante dei test di regressione.