UNIVERSITÀ DEGLI STUDI DI SALERNO

Dipartimento di Informatica

Corso di Ingegneria, Gestione ed Evoluzione del Software

Master Test Plan

TEAM MEMBER

Donia Daniele - 0522501575

La Marca Antonio - 0522501557

Somma Pasquale - 0522501543

Indice

CAPITOLO 1

Obiettivo e contesto

- **Nome del prodotto:** Perseverance
- **Descrizione del prodotto:** Si tratta di un tool sviluppato in Python che è in grado di minare repository tramite l'utilizzo di Pydriller, a partire da un dataset di progetti open source vulnerabili. Raccoglie i file java che hanno subito modifiche in commit che correggono delle vulnerabilità, ne estrae metriche del software, effettua mining del codice sorgente e analisi statica automatica.
- **Obiettivi del testing:** Garantire la qualità del prodotto, verificarne la corretta funzionalità e comprendere meglio il suo funzionamento.

CAPITOLO 2

Approccio e strategie di testing

L'approccio utilizzato per la fase di testing sarà il Bottom-up, effettuando il testing prima su unità più piccole e poi man mano integrando progressivamente i componenti più complessi che utilizzano quelli già testati. Attraverso uno studio della documentazione presente e, soprattutto, un'analisi statica del codice, supportata dalla presenza di commenti molto descrittivi e da una buona nomenclatura dei metodi, siamo riusciti a comprendere in modo adeguato il funzionamento del tool. Questo ci ha permesso di risalire alle specifiche attese, ovvero ai comportamenti desiderati dei metodi. Abbiamo inoltre identificato gli oggetti con cui i metodi, e successivamente gli script che li contengono, interagiscono, nonché i relativi effetti. Sulla base di queste osservazioni, abbiamo deciso di adottare un approccio di testing **black-box** per tutti i livelli di verifica (unità, integrazione e sistema). Questo approccio ci ha consentito di concentrarci sugli obiettivi funzionali e sulla verifica della correttezza del comportamento, evitando di soffermarci sui dettagli implementativi, che avrebbero potuto risultare complessi. La strategia presa in considerazione sarà il **category partitioning**, utilizzato per generare casi di test suddividendo gli input in categorie significative, quindi combinando vari scenari per verificare come il sistema ottiene i file dalle repository scelte dall'utente e ne estrae le metriche. Utilizzeremo proprietà e selettori per ridurre il numero di casi di test, evitando di eseguire quelli non significativi o privi di utilità.

- **Test di Unità:** Il testing sarà condotto sulle funzioni presenti negli script, sulla base di:
  - Regular Expression/Patterns: i link alle repository hanno un determinato pattern da seguire per verificarne la correttezza. Cercheremo di fornire quanti più input che possano mettere in difficoltà il sistema in tal senso, fornendo link inutilizzabili o che possono essere mal interpretati.
  - Oggetti dell'ambiente: pochi metodi all'interno del sistema possiedono parametri su cui poter lavorare in fase di test, per cui utilizzeremo molto l'ambiente di lavoro come directory, esistenza o meno di file e il loro contenuto; dal momento che il tool segue percorsi fissi all'interno del sistema e dà per scontate molte situazioni senza effettuare adeguati controlli.
- **Test di integrazione:** Eseguiremo uno dopo l'altro tutti gli script appartenenti al medesimo modulo, visto che molto spesso l'output del precedente è l'input del successivo.
- **Test di sistema:** Dal momento che non vi è un vero e proprio sistema, ma vengono semplicemente eseguiti degli script in un determinato ordine, il test di sistema verrà effettuato al termine delle modifiche e alla creazione di un vero e proprio sistema che si basa sul tool.
- **Test di regressione:** Si utilizzerà l'approccio TEST-ALL in quanto l'esecuzione di tutti i test esistenti non supera i 10 minuti e ci consente di ritestare tutte le funzionalità del sistema. Cercheremo anche di confrontare i risultati ottenuti prima delle modifiche con quelli ottenuti in seguito, quando possibile.

CAPITOLO 3

Pass/Fail criteria

Il test si considera superato se l'output corrisponde all'output atteso presente nell'oracolo; il test si considera fallito se l'output non corrisponde all'output atteso presente nell'oracolo. Nel caso in cui verrà riscontrata una failure, bisognerà verificare a che tipo di fault è legata e successivamente si procederà alla sua correzione. Infine, sarà eseguita di nuovo la fase di testing per verificare che la modifica effettuata non abbia creato ulteriori failure.

CAPITOLO 4

Attività di testing e parti coinvolte

Il testing del sistema prevede le seguenti attività:

- creazione test case;
- esecuzione dei test;
- report dei test;
- gestione delle failure.

Verranno sottoposti a Test di Unità e successivamente a Test di Integrazione tutti gli script appartenenti ai moduli 'RepoMining', 'Text_Mining', 'mining_results_asa' e 'Union'. L'unico script escluso dal testing è `add_class_type.py` del modulo 'Software_Metrics' perchè è considerato esterno al contesto.

CAPITOLO 5

Strumenti per il testing

Per il testing delle unità e delle integrazioni utilizzeremo **pytest**, sfruttando lo strumento delle fixture per gestire in modo efficiente la configurazione dei test. La scelta di pytest è motivata dalla sua semplicità, dalla flessibilità e dalla possibilità di scrivere test chiari e scalabili. Per isolare i vari componenti del codice utilizzeremo inoltre i **mock** forniti dalla libreria unittest, che ci permettono di simulare dipendenze esterne e focalizzarci sul comportamento del codice in esame.

Per il testing di sistema, invece, useremo **Pywinauto** per interagire con l'applicazione Python che verrà sviluppata. Questa libreria è un insieme di moduli per automatizzare la Microsoft Windows GUI e ci consente di simulare interazioni utente sull'interfaccia grafica, verificando il funzionamento complessivo del sistema.