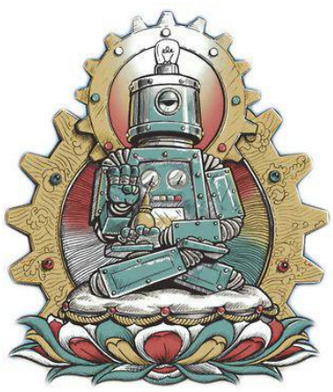
UNIVERSITÀ DEGLI STUDI DI SALERNO
CORSO DI INGEGNERIA DEL SOFTWARE

ASCETIC

Automated Code Smell Identification and Correction

TEST PLAN

VERSION 1.1



23/01/2019

Coordinatore Progetto:

Nome	**Matricola**
Manuel De Stefano	0522500633

Partecipanti:

Nome	**Matricola**
Amoriello Nicola	0512104742
Di Dario Dario	0512104758
Gambardella Michele Simone	0512104502
Iovane Francesco	0512104550
Pascucci Domenico	0512102950
Patierno Sara	0512103460

Revision History:

Data	**Versione**	**Descrizione**	**Autore**
17/12/2018	1.0	Prima Stesura	Manuel De Stefano
23/01/2019	1.1	Formalizzazione dei Casi di Test	Francesco Iovane

Indice

1 Introduzione

Lo scopo di questo documento è di pianificare l'attività di test del plugin **Ascetic** al fine di verificare se esistono differenze tra il comportamento atteso e il comportamento osservato. In questa attività andremo a rilevare gli eventuali errori prodotti all'interno del codice, per evitare che essi si presentino nel momento in cui il sistema verrà utilizzato dall'utente finale.

Le attività di test che sono state pianificate riguardano quelle parti del sistema che effettivamente sono state implementate (indicate nel documento di Object Design), tralasciando, almeno al momento della stesura di questa versione del documento, funzionalità indicate come di bassa e media priorità che sono state esclusivamente analizzate (le prime) o progettate ad alto livello (le seconde).

Oltre alla gestione dei test delle funzionalità, vengono anche pianificate le responsabilità del team e lo scheduling del test. La fase di testing è strettamente legata alle fasi ad essa precedenti: ogni documento, risultato delle differenti fasi di sviluppo, sarà un punto di partenza indispensabile per poter effettuare un testing corretto e adeguato.

2 Documenti Correlati

Il presente documento presenta una stretta correlazione con gli artefatti precedentemente prodotti, in quanto era già stato pianificato durante le fasi di produzione dei suddetti artefatti e quindi essi sono fondamentali per rilevare comportamenti inattesi durante la fase di testing qui descritta. Di seguito verranno riportate le relazioni tra il test plan e la documentazione precedente.

2.1 Relazioni con il *Requirement Analysis Document*

I test cases prodotti per il test di sistema avrà come specifiche i requisiti funzionali e non funzionali espressi nel RAD, mentre il comportamento descritto dai vari use cases fungerà da oracolo ai test.

2.2 Relazioni con il *System Design Document*

Nel documento di design di sistema, è stata scelta come architettura del sistema la cosidetta "three-tier architecture", che prevede la divisione del sistema in tre livelli: *Presentation*, *Application* e *Storage Layer*. Si pianifica che i test debbano rimanere quanto più fedeli possibile a tale divisione.

2.3 Relazioni con l' *Object Design Document*

In tale artefatto sono state stabilite le interfacce pubbliche dei vari moduli che compongono il sistema. Il test di integrazione assumerà queste come specifiche per i casi di test.

3 Panoramica del Sistema

Il sistema presenta due fondamentali suddivisioni, come specificato nel *System Design Document*: la prima scompone il sistema nei tre layer *Presentation*, *Application* e *Storage*, e la seconda scompone l'*Application Layer* in moduli coesi che realizzano funzionalità correlate. Tali moduli vanno a realizzare le funzionalità principali del plugin, ovvero l'analisi e la correzione automatiche di *code smells*. Saranno proprio queste, dunque, le funzionalità oggetto di testing.

4 Funzionalità da Testare e da Non Testare

Di seguito sono elencate le funzionalità che saranno testate:

1. Ricerca Smell
2. Correzione Blob
3. Correzione Promiscuous Package
4. Correzione Feature Envy
5. Correzione Misplaced Class

Che corrispondono alle funzionalità che effettivamente verranno implementate durante questa fase.

5 Criteri Pass/Failed

I dati di input del test saranno suddivisi in classi di equivalenza, ovvero verranno raggruppati in insiemi dalle caratteristiche comuni, per i quali sarà sufficiente testare un solo elemento rappresentativo. Un input avrà superato un test se l'output risultante sarà quello atteso, cioè quello che è stato specificato dal membro del team che si occuperà del testing su tale test case. Il responsabile del testing, invece, conosce quale dovrebbe essere l'output corretto.

6 Approccio

Le tecniche di testing adottate riguarderanno inizialmente il testing di unità dei singoli componenti, in modo da testare nello specifico la correttezza di ciascuna unità. Seguirà il testing di integrazione, che focalizzerà l'attenzione principalmente sul test delle interfacce delle suddette unità. Infine verrà eseguito il testing di sistema, che vedrà come oggetto di testing l'intero sistema assemblato nei suoi componenti. Quest'ultimo servirà soprattutto a verificare che il sistema soddisfi le richieste del committente.

6.1 Testing di Unità

Durante questa fase, verranno ricercate le condizioni di fallimento, isolando i componenti ed usando, dove necessario, test driver e stub, cioè implementazioni parziali di componenti che dipendono o da cui dipendono le componenti da testare. La strategia utilizzata per il testing si baserà esclusivamente sulla tecnica Black-Box, che si focalizza sul comportamento Input/Output, ignorando la struttura interna della componente. Al fine di minimizzare il numero di test cases, i possibili input verranno partizionati in classi di equivalenza e per ogni classe verrà selezionato un test case.

Il testing di unità sarà completamente a carico dello sviluppatore dell'unità stessa, al fine di aumentare la produttività dello stesso e ridurre i tempi di testing. Lo sviluppatore dovrà, dunque, non solo provvedere all'esecuzione del test, ma anche alla preparazione della test suite, della specifica dei casi di test e alla stesura di un eventuale test incident report e di un test execution report. Il primo sarà necessario nel caso in cui si riscontrino anomalie. In esso verranno riportati il caso di test che ha prodotto l'anomalia, l'input che l'ha generata e l'eventuale correzione. Il secondo report, invece, sarà uno storico di tutti i test eseguiti sulla singola componente.

6.2 Testing d'Integrazione

In questa fase si procederà all'integrazione delle componenti di una funzionalità che verranno testate nel complesso attraverso una strategia Bottom-Up. Si passerà, poi, alla funzionalità successiva fino ad esaurire le funzionalità implementate. Quest'approccio mira principalmente a ridurre le dipendenze tra funzionalità differenti e a facilitare la ricerca di errori nelle interfacce di comunicazione tra sottosistemi.

6.3 Testing di Sistema

Lo scopo di questa fase di testing è quello di dimostrare che il sistema soddisfi effettivamente i requisiti richiesti e sia, quindi, pronto all'uso. Come per il testing di unità, si cercherà di testare le funzionalità più importanti per l'utente e quelle che hanno una maggiore probabilità di fallimento. Si noti che, come per il testing di unità, si procederà attraverso tecnica Black-Box.

7 Sospensione e Ripresa

7.1 Criteri di Sospensione

La fase di testing del sistema verrà sospesa quando si raggiungerà un compromesso tra qualità del prodotto e costi dell'attività di testing. Il testing verrà quindi portato avanti quanto più possibile nel tempo senza però rischiare di ritardare la consegna finale del progetto.

7.2 Criteri di Ripresa

Ogni qual volta verranno effettuate modifiche al sistema, sia che esse aggiungano funzionalità, sia che esse correggano difetti, la fase di testing verrà ripresa al fine di verificare che le componenti modificate (o aggiunte) non presentino anomalie. Verrà inoltre rieseguita tutta la test suite per assicurarsi che le modifiche non abbiano indirettamente alterato il comportamento di altre componenti (testing di regressione).

8 Materiale per il Testing

Per poter effettuare il testing sarà necessario unicamente un PC con una versione di Intellij IDEA installata. Poiché il software consiste in un plugin, il testing ad ogni livello verrà effettuato sfruttando gli strumenti messi a disposizione dalla stessa piattaforma Intellij.

9 Test Cases

1. **Analisi e ricerca Smell**

Parametri	Smell da cercare, Scope di ricerca
Smell analizzabile[SA]	1. Lo smell da ricercare è presente nella lista dei code smell presi in analisi dal sistema [property SA OK] 2. Lo smell da ricercare non è presente nella lista dei code smell presi in analisi dal sistema [error]
Insieme di package da analizzare[PA]	1. Non ci sono package all'interno del progetto [if SA OK][error] 2. C'è un sono package nel progetto [if SA OK][OK] 3. Ci sono N (con n>1) package nel progetto [if SA OK][OK]

Codice	Combinazione	Esito
TC 1.0	SA1,PA1	Errato
TC 1.1	SA2,PA1	Errato
TC 1.2	SA2,PA2	Errato
TC 1.3	SA2,PA3	Errato
TC 1.4	SA1,PA2	Corretto
TC 1.5	SA1,PA3	Corretto

2. **Correzione Misplaced Class**

Parametri	Classe da spostare, Package destinazione
Classe affetta[CA]	1. Esiste una classe affetta dallo smell "Misplaced Class" [property CA OK] 2. Non esiste una classe affetta dallo smell "Misplaced Class" [error]
Package destinazione[PD]	1.Il package di destinazione esiste [if CA OK][OK] 2.Il package di destinazione non esiste[if CA OK][error]

Codice	Combinazione	Esito
TC 2.0	CA1,PD2	Errato
TC 2.1	CA2,PD1	Errato
TC 2.2	CA2,PD2	Errato
TC 2.3	CA1,PD1	Corretto

3. **Correzione Feature Envy**

Parametri	Metodo da spostare, Classe Destinazione
Classe Affetta[AC]	1. Esiste un metodo affetto dallo smell "Feature Envy" [property AC OK] 2. Non esiste un metodo affetto dallo smell "Feature Envy" [error]
Classe Destinazione [CD]	1. La classe di destinazione esiste[if AC OK][OK] 2. La classe di destinazione non esiste [if AC OK][error]

Codice	Combinazione	Esito
TC 3.0	AC1,CD2	Errato
TC 3.1	AC2,CD1	Errato
TC 3.2	AC2,CD2	Errato
TC 3.3	AC1,CD1	Corretto

4. **Correzione Blob**

Parametri	Classe affetta, Prima classe splittata, Seconda classe splittata
Classe affetta[AB]	1. Esiste una classe affetta dallo smell "Blob" [property AB OK] 2. Non esiste una classe affetta dallo smell "Blob"[error]
Prima classe splittata[SPF]	1. Non esiste una classe col nome della nuova classe da creare [if AB OK][OK] 2. Esiste già una classe col nome della nuova classe da creare [if AB OK][error]
Seconda classe splittata [SPS]	1. Non esiste una classe col nome della seconda nuova classe da creare [if AB AND SPF OK][OK] 2. Esiste già una classe col nome della nuova classe da creare [if AB OK][error]

Codice	Combinazione	Esito
TC 4.0	AB1,SPF2,SPS1	Errato
TC 4.1	AB1,SPF2,SPS2	Errato
TC 4.2	AB1,SPF1,SPS2	Errato
TC 4.3	AB2,SPF1,SPS1	Errato
TC 4.4	AB2,SPF2,SPS1	Errato
TC 4.5	AB2,SPF1,SPS2	Errato
TC 4.6	AB2,SPF2,SPS2	Errato
TC 4.7	AB1,SPF1,SPS1	Corretto

5. **Correzione Promiscuous Package**

Parametro	Package affetto, Primo package splittato, Secondo package splittato
Package affetto [AP]	1. Esiste un package affetto dallo smell "Promiscuous Package" [property AP OK] 2. Non esiste un package affetto dallo smell "Promiscuous Package"[error]
Primo package splittato[PPF]	1. Non esiste un package col nome del nuovo package da creare [if AP OK][OK] 2. Esiste già un package col nome del nuovo package da creare [if AP OK][error]
Seconda classe splittata [PPS]	1. Non esiste un package col nome del secondo nuovo package da creare [if AP OK AND PPS OK][OK] 2. Esiste già un package col nome del nuovo package da creare [if AP OK][error]

Codice	Combinazione	Esito
TC 4.0	AP1,PPF2,PPS1	Errato
TC 4.1	AP1,PPF2,PPS2	Errato
TC 4.2	AP1,PPF1,PPS2	Errato
TC 4.3	AP2,PPF1,PPS1	Errato
TC 4.4	AP2,PPF2,PPS1	Errato
TC 4.5	AP2,PPF1,PPS2	Errato
TC 4.6	AP2,PPF2,PPS2	Errato
TC 4.7	AP1,PPF1,PPS1	Corretto

Per la specifica dei test cases di sistema si rimanda all'apposito documento di specifica dei casi di test di sistema e di integrazione.

10 Pianificazione del Testing

Il team per il testing (di integrazione e di sistema) deve essere composto da persone che hanno una completa e approfondita conoscenza del sistema e delle tecniche di testing con i documenti associati, quali *Test Plan* e *Test Case Specification*. Tali tecniche devono essere applicate nei tempi, nel budget e nei vincoli di qualità stabiliti. Il team dedicato al controllo della qualità è responsabile dell'attività di testing e quindi della ricerca di fault. La documentazione dei fault trovati è inviata agli sviluppatori per consentire la correzione del sistema. Il sistema revisionato è poi testato nuovamente non solo per verificare se gli errori trovati in precedenza sono stati eliminati ma soprattutto per verificare che non ne siano stati introdotti dei nuovi. L'attività di testing è fondamentale nello sviluppo di un sistema software in quanto la mancanza di tale attività o una cattiva interpretazione di essa può portare al completo fallimento del sistema. Data l'importanza del testing ne risulta fondamentale la schedulazione.

10.1 Determinazione dei Ruoli

Il team dedicato all'attività di testing di sistema sarà composto da: Le attività relative al testing di unità,come già detto in precedenza, saranno delegate agli stessi sviluppatori delle componenti, in modo tale da alleggerire il lavoro del team di testing, che potrà quindi dedicarsi ad un più attento lavoro di testing funzionale.

10.2 Determinazione dei Rischi

I rischi di un completo fallimento verranno minimizzati effettuando una pianificazione verticale delle attività di testing funzionale. Questo permetterà in caso di ritardi, dovuti ad una grande quantità di failure trovati, di rilasciare meno funzionalità del previsto, ma completamente testate. Inoltre tale pianificazione ridurrà notevolmente la produzione di driver e stub, evitando l'introduzione di nuovi errori, dovuti all'implementazione di tali componenti.

10.3 Organizzazione delle attività di testing

Le attività di testing verranno organizzate secondo uno schema che effettuerà una divisione funzionale di tipo verticale. In questo modo al termine di ogni attività si avrà una funzionalità completamente testata nei suoi livelli gerarchici. I vantaggi principali sono che in caso di ritardi dovuti al ritrovamento di numerosi failure il sistema verrà rilasciato con meno componenti, ma interamente testate e funzionanti.