Test Documentation

cASpeR su Maven

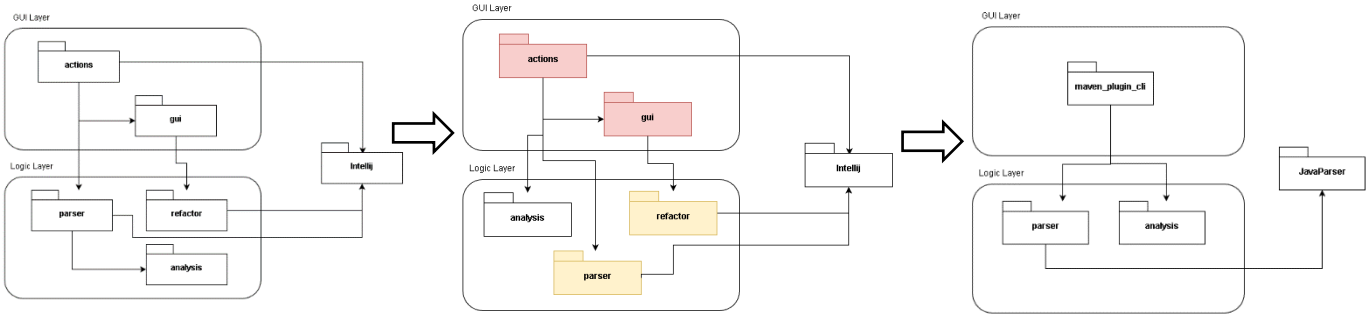
Riferimento	
Data	20/07/2022
Destinatario	Prof. De Lucia, Dott. Di Nucci, Dott. Manuel De Stefano, Dott. Emanuele Iannone
Presentato da	Lerose Ludovico, Milione Vincent
Approvato da	

Sommario

1. Introduzione e Planning

Lo scopo del documento è quello di illustrare la pianificazione delle attività di testing e le strategie di testing riguardanti l'applicazione cASpER reingegnerizzata come maven plugin.

Nelle sezioni di analisi e progettazione del documento di progetto sono state proposte e analizzate le varie soluzioni alle introduzioni delle modifiche. Infine, sono state decise le azioni da intraprendere e la vista del sistema finale, a seguito dell'introduzione di MR1 e MR2, è la seguente.



In particolare, notiamo che a seguito delle modifiche da eseguire:

- l parser verrà modificato ed isolato mediante un'operazione di extract class refactoring dei metodi responsabili di avviare il processo di analisi
- Il modulo "refactoring" non verrà trasferito, in quanto sarà implementato in un successivo incremento

L'obiettivo di queste attività di testing è assicurarsi che il plugin versione maven si comporti in maniera simile alla versione originale. In particolare, vogliamo che la parte responsabile dell'analisi statica comunichi gli stessi risultati della versione originale. A seguito delle informazioni recepite dal documento di progetto e dell'obiettivo illustrato possiamo dedurre che i due moduli che necessitano assolutamente di essere testati sono il parser e il nuovo modulo "maven_plugin_cli". Inoltre, è assolutamente necessario un test di sistema.

1.1 MR1

Data la restrizione dello scope di refactoring dovuta a MR2, le parti effettivamente re-fattorizzate sono nel modulo di analisi. Ricordiamo che MR1 si prepone di eliminare moduli "polluted". Per moduli "polluted" s'intende moduli con delle funzionalità esistenti nel codice, ma mai usate.

Come discusso nella sezione 3.1 e 2.1 del documento di progetto, le parti da re-fattorizzare sono code smell e le loro relative strategie. Grazie all'applicazione dello strategy pattern, rimuovere un code smell con le sue relative strategie o delle strategie in particolare è piuttosto semplice. In più le modifiche non impattano gli altri code smell, i loro algoritmi di detection e il loro uso nella fase di analisi statica. Bensi chi usa i code smell e le loro relative strategie vengono impattati dalla modifica.

Come accennato nelle parti finali della sezione 3.1 del documento di progetto, i cambiamenti si propagano in linea teorica verso il parser e l'interfaccia grafica. Per questo non è richiesto il testing di MR1:

- Durante le attività di reverse engineering, si è notato che il parser non avvia né l'analisi statica per l'individuazione dei nuovi code smell implementati né applica i nuovi algoritmi di history

implementati. Il codice responsabile è stato commentato. Non dovendo modificarlo, non è necessario testarlo.
- L'interfaccia grafica verrà completamente reingegnerizzata.

Quindi le modifiche rimangono locali al modulo di analisi e non impattano in nessun modo le componenti esistenti all'interno.

Pur essendo dotati di una test suite per il modulo di analisi, non è comunque necessario un test di regressione, ma lo si può fare per completezza. Inoltre, come vedremo nella sezione successiva, verrà comunque eseguito un test per il parser e a livello di sistema. In questi test potremmo verificare la presenza di malfunzionamenti dovuti all'estrazione di questi moduli "polluted".

1.2 MR2

Come si è visto dalle figure iniziali, l'intervento di reengineering coinvolge più layer del sistema.

I due approcci al testing a cui si è pensato per MR2 sono gli approcci bottom-up e sandwich. L'approccio top-down potrebbe avere senso siccome le modifiche sostanziali sono alla parte interattiva del tool. Si deve però considerare che l'uso della nuova libreria – Java parser – è in grado di stravolgere il funzionamento dell'intero sistema. Siccome tutta l'attività di analisi statica inizia a seguito del parsing del progetto software, l'approccio top down non può essere adottato. È necessario testare prima il modulo parser per verificare la presenza di malfunzionamenti in esso.

Dunque, non restano altro che i due approcci inizialmente indicati. L'approccio sandwich sarebbe l'ideale per i nostri scopi. Sia la logica del parser che la logica di presentazione potrebbero essere testate in isolamento. Dato che il modulo di analisi non ha bisogno di essere ritestato in isolamento, possiamo poi procedere direttamente con l'integrazione delle tre componenti ed eseguire un test di sistema complessivo.

Nell'approccio bottom-up, l'idea sarebbe quella di testare in isolamento la logica del parser e poi procedere con l'integrazione con la logica di presentazione. Come già accennato, il parser è responsabile per strutturare l'input per il modulo di analisi. Di conseguenza il modulo maven_plugin_cli non restituirebbe nulla in output se non viene integrato anche con la parte di analisi. Una volta testato in isolamento il modulo parser, si effettuerebbe immediatamente un test di sistema.

Bottom-Up		**Sandwich**	
Pro	Contro	Pro	Contro
Riduciamo i tempi di testing	Non viene testata la logica di presentazione in isolamento	Tutti i moduli introdotti e modificati vengono testati in isolamento	Maggiori tempi richiesti per il testing
	+ Rischi di malfunzionamento nella logica di presentazione	- Rischi di malfunzionamento nella logica di presentazione	

Bisogna tenere conto che l'effort per realizzare dei casi di test per il parser e per il test di sistema èalto per via della scarsa possibilità di automazione. Questi casi di test non esistono nella versione originale di cASpER. Di conseguenza, non solo bisognerebbe ideare i casi di test per la versione maven, ma bisognerebbe estrarre i nostri oracoli dal sistema originale per effettuare il confronto.

Dovendo limitare l'effort da impiegare considerando che i membri nel gruppo sono solo due, si è preferito l'approccio bottom-up. L'approccio sandwich avrebbe senso nel caso in cui siano richiesti più livelli di integrazione, in modo da non ritardare troppo il testing dei moduli di "coordinamento". Questo non è il

caso nostro. Subito dopo il testing del parser, possiamo testare almeno una volta la logica di presentazione a livello di sistema.

1.3 Test selection strategy

Di seguito riportiamo le strategie di selezione dei casi di test ai due livelli identificati. Quello unitario per il parser e quello a livello di sistema.

Testing parser

Come accennato verso la fine del paragrafo introduttivo, non abbiamo a disposizione nel tool originale un test per il parser. A seguito della sezione 4 del documento di progetto, si può concludere che del modulo parser è necessario verificare la presenza di errori nel metodo parse() della classe ProjectParser.java.

Il metodo parse è responsabile della costruzione dei beans nel package storage.beans ed è completamente dipendente dalla libreria Javaparser. Di conseguenza il testing funzionale del metodo parse() non può prescindere da questi elementi. Quindi parse() non verrà testato in isolamento, ma insieme a questi elementi.

Per affrontare il testing funzionale del parser si è deciso di applicare la tecnica black-box (quindi un approccio sistematico) del **category partition**, in quanto sembra essere la più flessibile tra tutte le strategie black-box che si sono studiate, in termini del dominio di input – i progetti software. L'applicazione della tecnica è vista nella sezione 2.1 di questo documento.

Uno dei problemi maggiori nel testare il parser è che la versione originale implementa due responsabilità, il parsing del progetto e l'avvio dell'analisi statica. Ciò rende l'estrazione automatica degli oracoli un po' più problematica. In quanto, nella versione Maven, siamo interessati solo a verificare il corretto funzionamento del parsing di un progetto.

In caso non sia possibile trovare un approccio automatico all'estrazione dell'oracolo, daremo il via alla loro estrazione manuale.

Testing di sistema

Seppure non esistano dei veri e propri test funzionali a livello di sistema, esistono però i test a livello di integrazione del modulo di analisi nel tool originale. Questi possono essere facilmente scalati a livello di sistema e usati con lo scopo di verificare il comportamento del tool maven in confronto a quello originale.

Per tanto si è deciso di prendere la test suite esistente ed adattarla a livello di sistema.

1.4 Testing Technologies and Procedures

Di seguito riportiamo le tecnologie e procedure da seguire all'esecuzione dei test case ai diversi livelli. Gli step indicano come generare il test report finale.

Testing parser

La realizzazione dei casi di test avverrà attraverso l'uso della tecnologia JUnit 4. Questo permetterà di automatizzare le attività di testing execution e la generazione del test report.

È previsto che i casi di test vengano eseguiti sull'IDE Intellij 2019.3 Ultimate Edition. Dato che il progetto è in Maven, l'esecuzione del test avverrà attraverso la fase del ciclo di vita chiamato "test". I risultati dei test saranno poi raccolti in un test report generato automaticamente dall'IDE in formato HTML. Identifichiamo e dettagliamo i seguenti step che devono essere presi per la fase di esecuzione e generazione del test report:

1) Aprire il progetto cASpER versione plugin maven con IDE Intellij 2019.3
    a. Se non è stato ancora importato il progetto, allora s'importa
2) Aprire la finestra laterale di Maven su Intellij
3) Aprire la tendina "Lifecycle"
4) Doppio click su opzione "test"
5) Al termine dell'esecuzione dei test, cliccare sull'icona dell'export dei risultati
6) Scegliere formato HTML e salvare con nome "Test Report – cASpER_plugin_maven.html"

Testing di sistema

Non avendo una conoscenza dei tool per effettuare test di sistema per plugins, il test sistema sarà condotto in modo più manuale.

Gli oracoli saranno estratti usando il task gradle "runIDE" nel progetto originale. Tale task consente di costruire un'istanza del programma Intellij 2019.3 con il plugin cASpER installato. Da qui proveremo ad eseguire il plugin sui progetti software che si deciderà di usare come input di un test case.

Di conseguenza, i risultati del test case dovranno essere riportati manualmente in un documento apposta.

2. Test Frames

Introduciamo in questa sezione il modo in cui è stato applicato category partition. In ogni sottosezione vedremo le categorie identificate per i parametri e loro relative scelte e vincoli. In seguito, saranno riportati i test frame conformi ai vincoli che si sono decisi di applicare. Non verrano specificati in questo documento i casi di test per il parser. Dato che quest'ultimo si è in grado di automatizzare, riportiamo la descrizione dei test case direttamente nella classe JUnit.

2.1 Test frame parser

Per il test unitario del parser si è pensato di utilizzare la tecnica di testing funzionale category partition.
Il parametro è unico ed è il progetto software Java da essere "parsato".

Precondizione: Progetto software Java Maven.

Postcondizione: Lista dei package bean istanziati con il nome del rispettivo package e i contenuti testuali delle classi contenute in esso.

Numero di packages
- 0 [property nopackages]
- 1 [property singlepackage]
- molti [property multipackage]

Numero di classi
- 0 [if nopackages]
- 1 [if singlepackage] [property singleclass]
- molti [if singlepackage] [if multipackage] [property manyclasses]

Buildabile
- si
- Errore lessicale presente del progetto [if singlepackage] if[singleclass]

- Errore sintattico presente in una classe del progetto [if singlepackage] [if singleclass]
- Errore semantico presente in una classe del progetto [if singlepackage] [if singleclass]

Numero di gerarchie
- 0
- 1 [if manyclasses]
- molti [if manyclasses]

Combinazioni

P_TC1: NumPackages[0], **NumClasses**[0], **Buildable**[si], **NumHierarchy**[0]

P_TC2: NumPackages[1], **NumClasses**[1], **Buildable**[si], **NumHierarchy**[0]

P_TC3: NumPackages[1], **NumClasses**[1], **Buildable**[errore lessicale], **NumHierarchy**[0]

P_TC4: NumPackages[1], **NumClasses**[1], **Buildable**[errore sintattico], **NumHierarchy**[0]

P_TC5: NumPackages[1], **NumClasses**[1], **Buildable**[errore semantico], **NumHierarchy**[0]

P_TC6: NumPackages[1], **NumClasses**[molti], **Buildable**[si], **NumHierarchy**[0]

P_TC7: NumPackages[1], **NumClasses**[molti], **Buildable**[si], **NumHierarchy**[1]

P_TC8: NumPackages[1], **NumClasses**[molti], **Buildable**[si], **NumHierarchy**[molti]

P_TC9: NumPackages[molti], **NumClasses**[molti], **Buildable**[si], **NumHierarchy**[0]

P_TC10: NumPackages[molti], **NumClasses**[molti], **Buildable**[si], **NumHierarchy**[1]

P_TC11: NumPackages[molti], **NumClasses**[molti], **Buildable**[si], **NumHierarchy**[molti]

2.2 Test frame di sistema

Come già detto, i test a livello di integrazione del modulo di analisi possono essere scalati a livello di sistema. Più precisamente si è deciso di riutilizzare il criterio di category partition alla base di quei test, tenendo conto che il parametro in input è unico ed è un progetto software Maven realizzato in Java. La category partition utilizzata presenta le seguenti categorie e scelte.

TheComponentIsSmelly

-Sì
-No

Soglia

-Default

-Non default

Combinazioni generali

-TheComponentIsSmelly[Sì], Soglia[Default]
-TheComponentIsSmelly[Sì], Soglia[Non default]
-TheComponentIsSmelly[No], Soglia[Non default]
-TheComponentIsSmelly[No], Soglia[default]

Le combinazioni generali sopracitate vanno applicate per ogni tipologia di smell e per ciascuno dei due approcci di analisi, ossia testuale e strutturale. Precisamente una certa componente è smelly o no se la metrica testuale o strutturale computata dal sottosistema di analisi rispettivamente è strettamente maggiore o minore uguale del valore di soglia definito. Dato che questi non possono essere automatizzati, riportiamo di seguito i test case a livello di sistema.

Test Case ID: SSFE_TC1	
Ambiente di Test	Soglia strutturale impostato a 0
Input	Progetto software: systemTests/Input/FeatureEnvyStrutturale
Oracolo	Il metodo getMobilePhoneNumber della classe Customer nel feature_envy package è affetta da Feature Envy Smell, presenta priorità alta e un risultato strutturale pari a 4

Test Case ID: SSFE_TC2	
Ambiente di Test	Soglia strutturale impostato a 3.0
Input	Progetto software: systemTests/Input/FeatureEnvyStrutturale
Oracolo	Il metodo getMobilePhoneNumber della classe Customer nel feature_envy package è affetta da Feature Envy Smell, presenta priorità alta e un risultato strutturale pari a 4

Test Case ID: SSFE_TC3	
Ambiente di Test	Soglia strutturale impostato a 4.0
Input	Progetto software: systemTests/Input/FeatureEnvyStrutturale
Oracolo	Il metodo getMobilePhoneNumber della classe Customer nel feature_envy package non è affetta da Feature Envy Smell

Test Case ID: SSFE_TC4	
Ambiente di Test	Soglia strutturale impostato a 0
Input	Progetto software: systemTests/Input/FeatureEnvyStrutturale

Oracolo	Il metodo getNumber della classe Phone nel feature_envy package non è affetta da Feature Envy Smell

Test Case ID: STFE_TC1	
Ambiente di Test	Soglia coseno impostato a 0
Input	Progetto software: systemTests/Input/FeatureEnvyTestuale
Oracolo	Il metodo getMobilePhoneNumber della classe Customer nel feature_envy package è affetta da Feature Envy Smell, presenta priorità media e un valore testuale pari a 0.029

Test Case ID: STFE_TC2	
Ambiente di Test	Soglia coseno impostato a 0.020
Input	Progetto software: systemTests/Input/FeatureEnvyTestuale
Oracolo	Il metodo getMobilePhoneNumber della classe Customer nel feature_envy package è affetta da Feature Envy Smell, presenta priorità media e un valore testuale pari a 0.029

Test Case ID: STFE_TC3	
Ambiente di Test	Soglia coseno impostato a 0.035
Input	Progetto software: systemTests/Input/FeatureEnvyTestuale
Oracolo	Il metodo getMobilePhoneNumber della classe Customer nel feature_envy package non è affetta da Feature Envy Smell

Test Case ID: STFE_TC4	
Ambiente di Test	Soglia coseno impostato a 0
Input	Progetto software: systemTests/Input/FeatureEnvyTestuale
Oracolo	Il metodo getNumber della classe Phone nel feature_envy package non è affetta da Feature Envy Smell

Test Case ID: SSB_TC1	
Ambiente di Test	Soglie strutturali impostate ai valori di default ELOC = 500 LCOM = 350 Feature Sum = 20
Input	Progetto software: systemTests/Input/BlobStrutturale
Oracolo	La classe Prodotto nel package blob è affetta da Blob smell, presenta priorità alta e valori strutturali pari a 10,21, ~538

Test Case ID: SSB_TC2	
Ambiente di Test	Soglie strutturali impostate ELOC = 45 LCOM = 5 Feature Sum = 5
Input	Progetto software: systemTests/Input/BlobStrutturale
Oracolo	La classe Prodotto nel package blob è affetta da Blob smell, presenta priorità alta e valori strutturali pari a 10,21, ~538

Test Case ID: SSB_TC3	
Ambiente di Test	Soglie strutturali impostate ELOC = 600 LCOM = 450 Feature Sum = 120
Input	Progetto software: systemTests/Input/BlobStrutturale
Oracolo	La classe Prodotto nel package blob non è affetta da Blob smell

Test Case ID: SSB_TC4

Ambiente di Test	Soglie strutturali impostate ai valori di default ELOC = 500 LCOM = 350 Feature Sum = 20
Input	Progetto software: systemTests/Input/BlobStrutturale
Oracolo	La classe BankAccount nel package blob non è affetta da Blob smell

Test Case ID: STB_TC1	
Ambiente di Test	Soglia coseno impostata a 0.5
Input	Progetto software: systemTests/Input/BlobTestuale
Oracolo	La classe Prodotto nel package blob è affetta da Blob smell, presenta priorità media e il valore testuale pari a 0.651

Test Case ID: STB_TC2	
Ambiente di Test	Soglia coseno impostata a 0.55
Input	Progetto software: systemTests/Input/BlobTestuale
Oracolo	La classe Prodotto nel package blob è affetta da Blob smell, presenta priorità media e il valore testuale pari a 0.651

Test Case ID: STB_TC3	
Ambiente di Test	Soglia coseno impostato a 0.652
Input	Progetto software: systemTests/Input/BlobTestuale
Oracolo	La classe Prodotto nel package blob non è affetta da Blob smell

Test Case ID: STB_TC4

Ambiente di Test	Soglia coseno impostata a 0.5
Input	Progetto software: systemTests/Input/BlobTestuale
Oracolo	La classe Bankaccount nel package blob non è affetta da Blob smell

Test Case ID: SSPP_TC1	
Ambiente di Test	Soglia strutturale impostate: Intraconnectivity: 0.5 Interconnectivity: 0.5
Input	Progetto software: systemTests/Input/PromiscousStrutturale
Oracolo	Il package con nome "promiscous_package.first" è affetto da Promiscuos Package smell, presenta priorità urgente e valori strutturali pari a 1,0.

Test Case ID: SSPP_TC2	
Ambiente di Test	Soglia strutturale impostate: Intraconnectivity: 1.0 Interconnectivity: -0.1
Input	Progetto software: systemTests/Input/PromiscousStrutturale
Oracolo	Il package con nome "promiscous_package.first" è affetto da Promiscuos Package smell, presenta priorità urgente e valori strutturali pari a 1,0.

Test Case ID: SSPP_TC3	
Ambiente di Test	Soglia strutturale impostate: Intraconnectivity: 1.0 Interconnectivity: 0.0
Input	Progetto software: systemTests/Input/PromiscousStrutturale
Oracolo	Il package con nome "promiscous_package.first" non è affetto da Promiscuos Package smell

Test Case ID: SSPP_TC4	
Ambiente di Test	Soglia strutturale impostate: Intraconnectivity: 0.5 Interconnectivity: 0.5
Input	Progetto software: systemTests/Input/PromiscousStrutturale
Oracolo	Il package con nome “promiscous_package.second” non è affetto da Promiscuos Package smell

Test Case ID: STPP_TC1	
Ambiente di Test	Soglia coseno impostata a 0.5
Input	Progetto software: systemTests/Input/PromiscousTestuale
Oracolo	Il package con nome “promiscous_package.first” è affetto da Promiscuos Package smell, presenta priorità media e ha un valore testuale pari a 0.566

Test Case ID: STPP_TC2	
Ambiente di Test	Soglia coseno impostata a 0.55
Input	Progetto software: systemTests/Input/PromiscousTestuale
Oracolo	Il package con nome “promiscous_package.first” è affetto da Promiscuos Package smell, presenta priorità media e ha un valore testuale pari a 0.566

Test Case ID: STPP_TC3	
Ambiente di Test	Soglia coseno impostata a 0.67
Input	Progetto software: systemTests/Input/PromiscousTestuale

Oracolo	Il package con nome “promiscous_package.fisrt” non è affetto da Promiscuos Package smell

Test Case ID: STPP_TC4	
Ambiente di Test	Soglia coseno impostata a 0.5
Input	Progetto software: systemTests/Input/PromiscousTestuale
Oracolo	Il package con nome “promiscous_package.second” non è affetto da Promiscuos Package smell

Test Case ID: SSMC_TC1	
Ambiente di Test	Soglia strutturale impostata a 0
Input	Progetto software: systemTests/Input/MisplacedStrutturale
Oracolo	La classe Cliente nel primo package di misplaced è affetta da Misplaced Class smell, presenta priorità alta e ha un valore strutturale pari a 2

Test Case ID: SSMC_TC2	
Ambiente di Test	Soglia strutturale impostata a 1.0
Input	Progetto software: systemTests/Input/MisplacedStrutturale
Oracolo	La classe Cliente nel primo package di misplaced è affetta da Misplaced Class smell, presenta priorità alta e ha un valore strutturale pari a 2

Test Case ID: SSMC_TC3	
Ambiente di Test	Soglia strutturale impostata a 3.0
Input	Progetto software: systemTests/Input/MisplacedStrutturale
Oracolo	La classe Cliente nel primo package di misplaced non è affetta da Misplaced Class smell

Test Case ID: SSMC_TC4	
Ambiente di Test	Soglia strutturale impostata a 0
Input	Progetto software: systemTests/Input/MisplacedStrutturale
Oracolo	La classe BankAccount nel primo package di misplaced non è affetta da Misplaced Class smell

Test Case ID: STMC_TC1	
Ambiente di Test	Soglia coseno impostata a 0
Input	Progetto software: systemTests/Input/MisplacedTestuale
Oracolo	La classe BankAccount nel secondo package di misplaced è affetta da Misplaced Class smell, presenta priorità media e ha un valore testuale pari a 0.334

Test Case ID: STMC_TC2	
Ambiente di Test	Soglia coseno impostata a 0.329
Input	Progetto software: systemTests/Input/MisplacedTestuale
Oracolo	La classe BankAccount nel secondo package di misplaced è affetta da Misplaced Class smell, presenta priorità media e ha un valore testuale pari a 0.334

Test Case ID: STMC_TC3	
Ambiente di Test	Soglia coseno impostata a 0.349
Input	Progetto software: systemTests/Input/MisplacedTestuale
Oracolo	La classe BankAccount nel secondo package di misplaced non è affetta da Misplaced Class smell

Test Case ID: STMC_TC4	
Ambiente di Test	Soglia coseno impostata a 0
Input	Progetto software: systemTests/Input/MisplacedTestuale
Oracolo	La classe Ristorante nel secondo package di misplaced non è affetta da Misplaced Class smell