**Università Degli Studi Di Salerno**

**Dipartimento di Informatica**

**Corso di Laurea Magistrale in Softaware Engeneering and IT Mangament**

****

INGEGNERIA, GESTIONE ED EVOLUZIONE DEL

SOFTWARE

ANNO ACCADEMICO 2021/22

**Evoluzione cASpER**

**CR\_01-CR\_02 Test Plan**

**Obiettivi del Testing**

Lo scopo del testing è quello di individuare la presenza di faults all’interno del sistema e condurre attività di debug. Il testing ha due “parametri” fondamentali, che sono gli input e l’oracolo, ovvero il risultato che vogliamo ottenere. A valle di ciò ricordiamo che, il testing ha successo nel caso in cui, dato un input al sistema, il risultato ottenuto è diverso dal valore che ci aspettiamo, ovvero dall’oracolo. Il sistema cASpER è un plugin per l’ambiente di sviluppo IntelliJ IDEA per il rilevamento e i refactoring dei code smell.

Prima di eseguire qualsiasi intervento di manutenzione verrano eseguiti i test di regressione dei vari test esistenti. Dopo il testing di regressione iniziale, che ci assicura che il plugin funzioni correttamente, si iniziano a sviluppare le CR descritte. Dopo ogni task portato a termine, si passa poi al testing della CR e in caso di fault si aggiusta la classe di produzione fino a quando il risultato del testing sia uguale a quello dell’oracolo.

Dopo il testing delle CR, viene rieseguito il testing di regressione, per avere la certezza che la nuova modifica effettuata non abbia intaccato il funzionamento di tutto il plugin.

**Test Strategy**



**Approccio**

La fase di testing si compone di tre attività: una prima fase si occuperà di trovare errori in una singola componente; avrà come compito quella di testare le funzionalità nate dall’integrazione dei vari sottosistemi e per ultimo andremo a testare l’intero sistema assemblato.

1. **Testing di unita**

Durante tale fase di testing verranno testate singolarmente le componenti al fine di evidenziare gli errori.

Per il testing di unità si utilizza la tecnica di “Black-Box testing” che si focalizza sul comportamento di I/O, senza preoccuparsi della struttura interna della componente.

Se per ogni dato input, siamo in grado di prevedere l’output, allora l’unità supera il test.

Nel momento in cui utilizzando solo la tecnica di “Black-Box testing” non otteniamo che il 100% dei branch sviluppati sia testato, integreremo anche la tecnica di “White-Box-testing”.

1. **Testing di integrazione**

Una volta che sono stati rilevati i bug per una singola componente e riparati, le componenti sono pronte per essere integrate in sottosistemi più grandi.

Pertanto dopo aver testato singolarmente le componenti del sistema, possiamo procedere a testarne le integrazioni.

Per il testing d’integrazione si utilizza la strategia: “bottom-up”.

I sottosistemi al livello più basso della gerarchia sono testati individualmente.

I successivi sottosistemi ad essere testati sono quelli che chiamano i sottosistemi testati in precedenza. Si ripete quest’ultimo passo finché tutti i sottosistemi non sono stati testati

1. **Testing di sistema**

Per il testing di sistema non si utilizzerà nessun tool, ma si procederà ad eseguire manualmente il plugin, con lo scopo di verificare il corretto funzionamento.

**Unit Test Plan**

Le CR ricevute implicano l’aggiunta di due nuove funzionalità per cASpER:

1. Il sistema dovrà supportare il rilevamento del code smell Spaghetti Code
2. Il sistema dovrà supportare il rilevamento del code smell SwissArmyKnife

Tuttavia per l’implementazione di tali funzionalità dovrà essere modificato il comportamento del parser in modo da mantenere traccia delle interfacce implementate e le classi estese di un oggetto PsiClass. Pertanto si ritiene opportuno testare anche il parser con lo scopo di verificare che le modifiche effettuate siano corrette.

Essendo che per il testing dei nuovi detector introdotti sarà necessario un oggetto ClassBean, abbiamo deciso di utilzzare la libreria *LightJavaCodeInsightFixtureTestCase* per evitare di dover scrivere manualmente tali classi.

Tale libreria crea un progetto fittizio su delle classi target scelte dal tester.

Abbiamo pensato di creare delle classi di test su cui creare un progetto fittizio e tramite il parser ottenere l’oggetto ClassBean di tali classi. L’oggetto creato poi sarà usato come input per il testing dei Detector.

Nei test già presenti i ClassBean vengono costruiti manualmente, ma riteniamo che tale approccio sia troppo costoso e con l’utilizzo della libreria *LightJavaCodeInsightFixtureTestCase*  riusciamo ad ottenere lo stesso risultato. Tuttavia e bene precisare che per la creazione degli oggetti ClassBean è richiesto l’utilizzo del parser, quindi bisogna essere sicuri che quest’ultimo funzioni correttamente. Il corretto funzionamento del parser è stato riscontrato tramite il test di regressione.

1. **Tools Utilizzati**

* JUnit: framework per il testing
* Mockito: l’isolamento dei test di unità può essere ottenuto utilizzando oggetti che simulano il comportamento degli oggetti reali. Mockito fornisce questa possibilità.
* Jacoco: libreria che consente la di generare dei report sulla Code Coverage del testing.

1. **Criteri pass/fail**

|  |  |
| --- | --- |
| Fail | Il test non ha trovato nessun fault |
| Pass | Il test ha trovato un fault |

1. **Sospensione e Terminazione**

|  |  |
| --- | --- |
| Sospensione | Il test e sospeso se almeno il 10% dei casi di test riportano errori: in queste condizioni, il team deve provvedere a correggere i fault prima di procedere con l’implementazione o il testing di nuove funzionalità |
| Terminazione | Il test si considera terminato, quando la totalità dei casi di test somministrati al sistema riporta esito negativo.  La suddetta condizione sussiste solo se il 100% dei branch sviluppati viene ricoperto in questa fase |

1. **Test Case Selection**

Per la selezione dei casi di test abbiamo seguito la tecnica del “Category Partition”.

**Test Item**

|  |  |  |
| --- | --- | --- |
| ID | Class | Caratteristica da testare |
| Parser\_Unit | PsiParser.java | Dato in input un progetto, per ogni classe estratta tiene traccia delle interfacce implementate e delle classi estese? |
| SpaghettiCode\_Unit | SpaghettiCodeSmell.java | Data in input una classe, identifica correttamente se la classe è affetta dal code smell Spaghetti Code? |
| SwissArmyKnife\_Unit | SwissArmyKnifeSmell.java | Data in input una classe, identifica correttamente se la classe è affetta dal code smell SwissArmyKnife? |

* **Parser\_Unit**

Avendo modificato il parser in modo che durante la conversione di un oggetto PsiClass in ClassBean si tiene traccia delle interfacci implementate e delle classi estese. Testiamo quindi tale funzionalità.

|  |  |  |
| --- | --- | --- |
| **Parser\_Unit** | | |
| **Parametri:** | Progetto Java | |
| **Oggetti dell’ambiente:** | - | |
| **Categorie:** | [C\_1] Progetto Java contenente una classe | |
| **Scelte [C\_1]:** | [SP\_1] Progetto Java con una classe che non implementano nessuna interfaccia | |
| [SP\_2] Progetto Java con una classe che implementa un interfaccia | |
| [SP\_3] Progetto Java con una classe che implementano più di una interfaccia | |
| [SP\_4] Progetto Java con una classe che non estendono nessuna classe | |
| [SP\_5] Progetto Java con una classe che estende una classe | |
|  | | |
| **Codice** | **Test Frame** | **Oracolo** |
| testParserExtractClassImplementate1 | SP\_1 | Il numero di interfacce implementate dalla classe è uguale a 0 |
| testParserExtractClassImplementate2 | SP\_2 | Il numero di interfacce implementate dalla classe è uguale 1 |
| testParserExtractClassImplementate3 | SP\_3 | Il numero di interfacce implementate dalla classe è maggiore di 1 |
| testParserExtractClasseEstesa1 | SP\_4 | Il numero di classi estese dalla classe è uguale a 0 |
| testParserExtractClasseEstesa2 | SP\_5 | Il numero di classi estese dalla classe è uguale a 1 |

* **SpaghettiCode\_Unit**

|  |  |  |
| --- | --- | --- |
| **SpaghettiCode\_Unity** | | |
| **Parametri:** | Classe Java | |
| **Oggetti dell’ambiente:** | - | |
| **Categorie:** | [CJ\_1] Classe Java | |
| **Scelte [CJ\_1]:** | [SC\_1] Classe Java non affetta da spaghetti code | |
| [SC\_2] Classe Java affetta da spaghetti code | |
|  | | |
| **Codice** | **Test Frame** | **Oracolo** |
| StructuralSpaghettiCodeStrategyTest.testNotIsSmelly | SC\_1 | “False” la classe non è affetta dallo smell spaghetti code |
| StructuralSpaghettiCodeStrategyTest.testIsSmelly | SC\_2 | “True” la classe è affetta dallo smell spaghetti code |

* **SwissArmyKnife\_Unit**

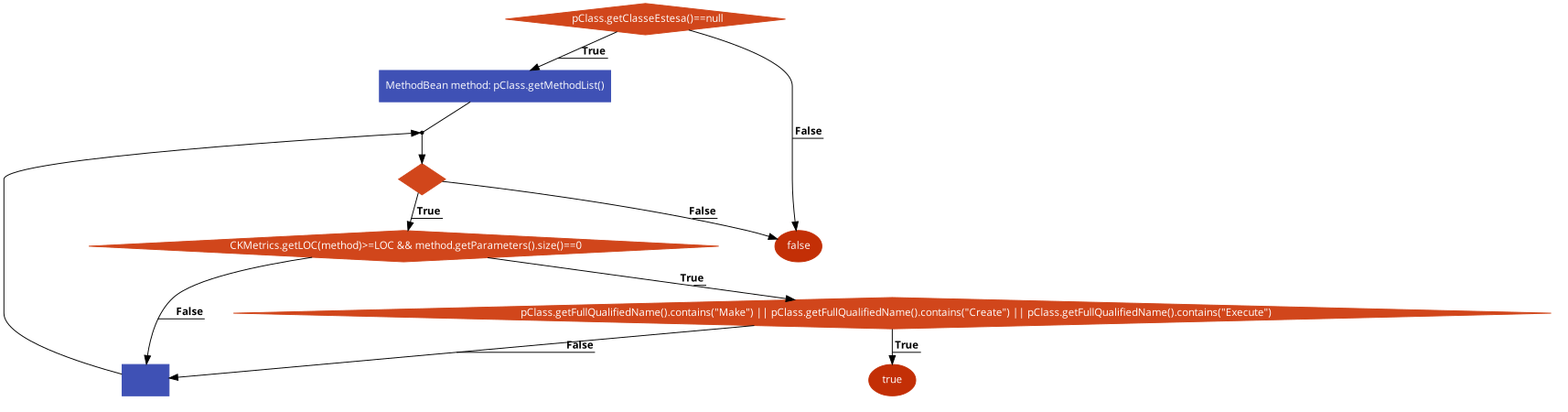
|  |  |  |
| --- | --- | --- |
| **SwissArmyKnife\_Unity** | | |
| **Parametri:** | Classe Java | |
| **Oggetti dell’ambiente:** | - | |
| **Categorie:** | [CJ\_1] Classe Java | |
| **Scelte [CJ\_1]:** | [SC2\_1] Classe Java non affetta da swiss army knife | |
| [SC2\_2] Classe Java affetta da swiss army knife | |
|  | | |
| **Codice** | **Test Frame** | **Oracolo** |
| StructuralSwissArmyKnifeStrategyTest .testNotIsSmelly | SC2\_1 | “False” la classe non è affetta dallo smell swiss army knife |
| StructuralSwissArmyKnifeStrategyTest .testIsSmelly | SC2\_2 | “True” la classe è affetta dallo smell swiss army knife |

* **Control Flow Graph**

Di seguito sono riportati i CFG dei detector che saranno utilizzati per identificare quali condizioni soddisfare al fine di incrementare la coverage. I seguenti CFG saranno confrontati con il report di jacoco. Per ogni branch non evidenziato nel report di jacoco sarà creato un test case che copre quel branch, andando a soddisfare la condizione necessaria affinché quel branch venga coperto.

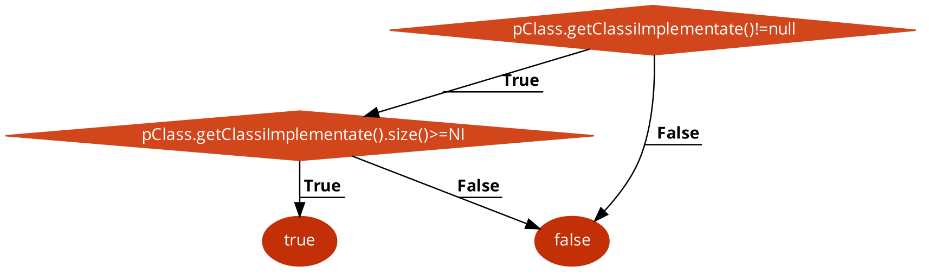
I seguenti CFG sono stati generati dal tool Code2Flow.

* **SpaghettiCode**



Documentazione generale\Testing\_CR\_01-CR\_02\Immagini\CFG\_SpaghettiCode

* **SwissArmyKnife**



Documentazione generale\Testing\_CR\_01-CR\_02\Immagini\CFG\_SwissArmyKnife

**Integration Test Plan**

Dopo aver testato con il test d’unita che sia le modifiche effettuate al parsere e i detector implementati funzionano correttamente, siamo passati all’eseguire il test d’integrazione tra queste due componenti.

1. **Criteri pass/fail**

|  |  |
| --- | --- |
| Fail | Il test non ha trovato nessun fault |
| Pass | Il test ha trovato un fault |

1. **Test Case Selection**

**Test Item**

|  |  |  |
| --- | --- | --- |
| ID | Class | Caratteristica da testare |
| Parser\_Unit&&SpagettiCode\_Unit Itegration | PsiParser.java  SpaghettiCodeSmell.java | Dato in input un progetto, l’analisi eseguita dal parser riesce a rilevare il code smell SpaghettiCode? |
| Parser\_Unit&&SwissArmyKnife\_Unit Itegration | PsiParser.java  SwissArmyKnifeSmell.java | Dato in input un progetto, l’analisi eseguita dal parser riesce a rilevare il code smell SwissArmyKnife? |

|  |  |  |
| --- | --- | --- |
| **Parser\_Unit&&SpagettiCode\_Unit Itegration** | | |
| **Parametri:** | Progetto Java | |
| **Oggetti dell’ambiente:** | - | |
| **Categorie:** | [PJ\_1] Progetto Java | |
| **Scelte [PJ\_1]:** | [SCPJ\_1] Progetto Java con una classe | |
| [SCPJ\_2] Progetto Java vuoto | |
|  | | |
| **Parametri:** | ClasseBean | |
| **Oggetti dell’ambiente:** | - | |
| **Categorie:** | [CLSJ\_1] ClasseBean | |
| **Scelte [CLSJ\_1]:** | [SPC\_1] ClasseBean affetta dallo smell SpaghettiCode [if SCPJ\_1] | |
| [SPC\_2] ClasseBean non affetta dallo smell SpaghettiCode [if SCPJ \_1] | |
|  | | |
| **Codice** | **Test Frame** | **Oracolo** |
| testIntegrationEmpityProject | SCPJ\_2 | Progetto vuoto |
| testIntegrationSpaghettiCodeIsSmelly | SCPJ\_1, SPC\_1 | “True” la classe è affetta dallo smell Spaghetti Code |
| testIntegrationSpaghettiCodeNotIsSmelly | SCPJ\_1, SPC\_2 | “False” la classe è affetta dallo smell Spaghetti Code |

|  |  |  |
| --- | --- | --- |
| **Parser\_Unit&&SwissArmyKnife\_Unit Itegration** | | |
| **Parametri:** | Progetto Java | |
| **Oggetti dell’ambiente:** | - | |
| **Categorie:** | [PJ\_1] Progetto Java | |
| **Scelte [PJ\_1]:** | [SCPJ\_1] Progetto Java con una classe | |
| [SCPJ\_2] Progetto Java vuoto | |
|  | | |
| **Parametri:** | ClasseBean | |
| **Oggetti dell’ambiente:** | - | |
| **Categorie:** | [CLSJ\_1] ClasseBean | |
| **Scelte [CLSJ\_1]:** | [SWSS\_1] ClasseBean affetta dallo SwissArmyKnife [if SCPJ\_1] | |
| [SWSS \_2] ClasseBean non affetta dallo smell SwissArmyKnife [if SCPJ \_1] | |
|  | | |
| **Codice** | **Test Frame** | **Oracolo** |
| testIntegrationEmpityProject | SCPJ\_2 | Progetto vuoto |
| testIntegrationSwissArmyKnifeIsSmelly | SCPJ\_1, SWSS \_1 | “True” la classe è affetta dallo smell SwissArmyKnife |
| testIntegrationSwissArmyKnifeNotIsSmelly | SCPJ\_1, SWSS \_2 | “False” la classe è affetta dallo smell SwissArmyKnife |

**System Test Plan**

Con il system test verrà eseguito i tool manualmente su un progetto java in cui sono presenti classi affette da spaghetti code e swissarmyknife. Verranno testati solamente i nuovi requisiti implementati, in quanto le modifiche apportate incideranno solamente su quello. Per osservare il corretto funzionamento del intero tool si utilizzerà il test di regressione post implementazione.

1. **Criteri pass/fail**

|  |  |
| --- | --- |
| Fail | Il test non ha trovato nessun fault |
| Pass | Il test ha trovato un fault |

1. **Test Case Selection**

|  |  |  |
| --- | --- | --- |
| ID | Class | Caratteristica da testare |
| SystemTestSpaghettiCode | PsiParser.java  SpaghettiCodeSmell.java  CheckProjectPaga.java  SpaghettiCodePage.java | Eseguito il plugin su un progetto contenente una classe affetta da spaghetti code, il plugin rileva tale smell? |
| SystemTestSwissArmyKnife | PsiParser.java  SwissArmyKnifeSmell.java  CheckProjectPaga.java  SwissArmyKnifePage.java | Eseguito il plugin su un progetto contenente una classe affetta da swiss army knife, il plugin rileva tale smell? |

|  |  |  |  |
| --- | --- | --- | --- |
| **SystemTestSpaghettiCode** | | | |
| **Parametri:** | Progetto Java | | |
| **Oggetti dell’ambiente:** | - | | |
| **Categorie:** | [PJ\_1] Progetto Java | | |
| **Scelte [PJ\_1]:** | [SPC\_1] Progetto Java con una classe affetta da Spaghetti Code | | |
| [SPC \_2] Progetto Java non contenente una classe affetta da Spaghetti Code | | |
|  | | | |
| **Codice** | | **Test Frame** | **Oracolo** |
| SystemTestSPC\_1 | | SPC\_1 | Nella lista degli smell presenti nel progetto analizzato è presente lo smell Spaghetti Code |
| SystemTestSPC\_2 | | SPC \_2 | Nella lista degli smell presenti nel progetto analizzato non è presente lo smell Spaghetti Code |

|  |  |  |  |
| --- | --- | --- | --- |
| **SystemTestSwissArmyKnife** | | | |
| **Parametri:** | Progetto Java | | |
| **Oggetti dell’ambiente:** | - | | |
| **Categorie:** | [PJ\_1] Progetto Java | | |
| **Scelte [PJ\_1]:** | [SWISS\_1] Progetto Java con una classe affetta da Spaghetti Code | | |
| [SWISS \_2] Progetto Java non contenente una classe affetta da Spaghetti Code | | |
|  | | | |
| **Codice** | | **Test Frame** | **Oracolo** |
| SystemTest\_SWISS\_1 | | SWISS\_1 | Nella lista degli smell presenti nel progetto analizzato è presente lo smell SwissArmyKnife |
| SystemTest\_SWISS\_2 | | SWISS\_2 | Nella lista degli smell presenti nel progetto analizzato non è presente lo smell SwissArmyKnife |