Università degli Studi di Salerno

Dipartimento di Informatica

A.A. 2021/22

cASpER

Supporto per nuove versioni di IntelliJ IDEA e aggiunta strategia di Extract Class Refactoring basata sulla Teoria dei Giochi

Testing post-modifica

Marco Calenda
0522501165

prof. **Andrea De Lucia**
prof. **Dario Di Nucci**

tutor **Manuel De Stefano**

Indice

Introduzione

Il seguente documento è incentrato sulla progettazione e realizzazione dei casi di test relativi alle modifiche apportate con CR_01 e CR_02. Viste le limitate modifiche, in termini di classi introdotte e nuove dipendenze create, non è stato necessario progettare un testing scalabile per l'intero sistema quindi è stato seguito un approccio white-box (strutturale) puntando a verificare la correttezza delle operazioni a livello di metodo e ad ottenere una buona copertura.

I test saranno poi eseguiti con il task di Gradle "test" e sarà esportato il report di esecuzione in formato HTML tramite le funzionalità dell'IDE IntelliJ IDEA. Infine, sarà eseguito il test di regressione per assicurarsi che le modifiche apportate non abbiano introdotto malfunzionamenti al resto del sistema.

Pianificazione Testing

In questa sezione è descritta la pianificazione delle attività di testing riguardanti le modifiche introdotte da CR_01 e CR_02.

CR_01

Considerata la natura adattiva di CR_01, non sono stati aggiunti casi di test funzionali rispetto a quelli già presenti. In relazione anche al limitato impatto delle modifiche (descritto nella sezione di Impact Analysis del documento "Analisi, progettazione e realizzazione delle CR"), per testare la bontà delle modifiche si è eseguito un test di sistema per verificare il requisito "La build del plugin deve avvenire con successo". È stato utilizzato il task "build" fornito da Gradle che comprende ulteriori sotto task.

ID	ST_01
Precondizioni	
Step	1*:* lo sviluppatore esegue il task "clean" di Gradle 2*:* lo sviluppatore esegue il task "build" di Gradle
Oracolo	La build del plugin avviene con successo
Note	Sono accettati eventuali warning

CR_02

L'unica funzionalità esterna (definita come requisito visibile all'utente) è:

- ER_01: "Il sistema deve essere in grado di applicare un'operazione di splitting basata sulla Teoria dei Giochi".

Testare interamente questa funzionalità risulterebbe molto complesso, infatti due vincoli principali violano alcune caratteristiche FIRST:

- non-determinismo: l'euristica basata su LDA, applicando un algoritmo di machine learning, può fornire risultati diversi con lo stesso input e questo viola la caratteristica "Repeatable". Un test case che verifica un algoritmo di machine learning necessita di un'attenta progettazione per evitare di produrre un flaky test.
- tempo: l'intero processo di splitting con Teoria dei Giochi (comprensivo di training del modello LDA e algoritmo iterativo ECR) risulta essere molto più lento del processo tradizionale e questo viola la caratteristica "Fast".

Per questo, è stata ridotta la granularità e sono state testate le funzionalità interne (non visibili all'utente) a livello di metodo ovvero:

- IR_01: "Il sistema deve calcolare la Jaccard Similarity tra due liste di stringhe"
- IR_02: "Il sistema deve normalizzare il contenuto testuale di una classe"
- IR_03: "Il sistema deve permettere il merge tra liste di stringhe che hanno una Jaccard Similarity maggiore di una soglia"
- IR_04: "Il sistema deve calcolare i payoffs data una lista di possibili scelte ed una matrice MethodByMethod"
- IR_05: "Il sistema deve trovare l'Equilibrio di Nash (o più di uno) data una matrice di payoffs"

È stato costruito il dependency graph (Figura 1) per organizzare un test d'integrazione bottom-up preceduto da un test di unità delle componenti terminali; utilizzando un approccio di testing white-box (strutturale), sono state verificate tutte le singole operazioni (*i.e.* intra-method testing) coinvolte nell'operazione di splitting in modo isolato, massimizzando lo Statement Coverage ed il Branch Coverage delle classi InputFinder e PayoffMatrix.

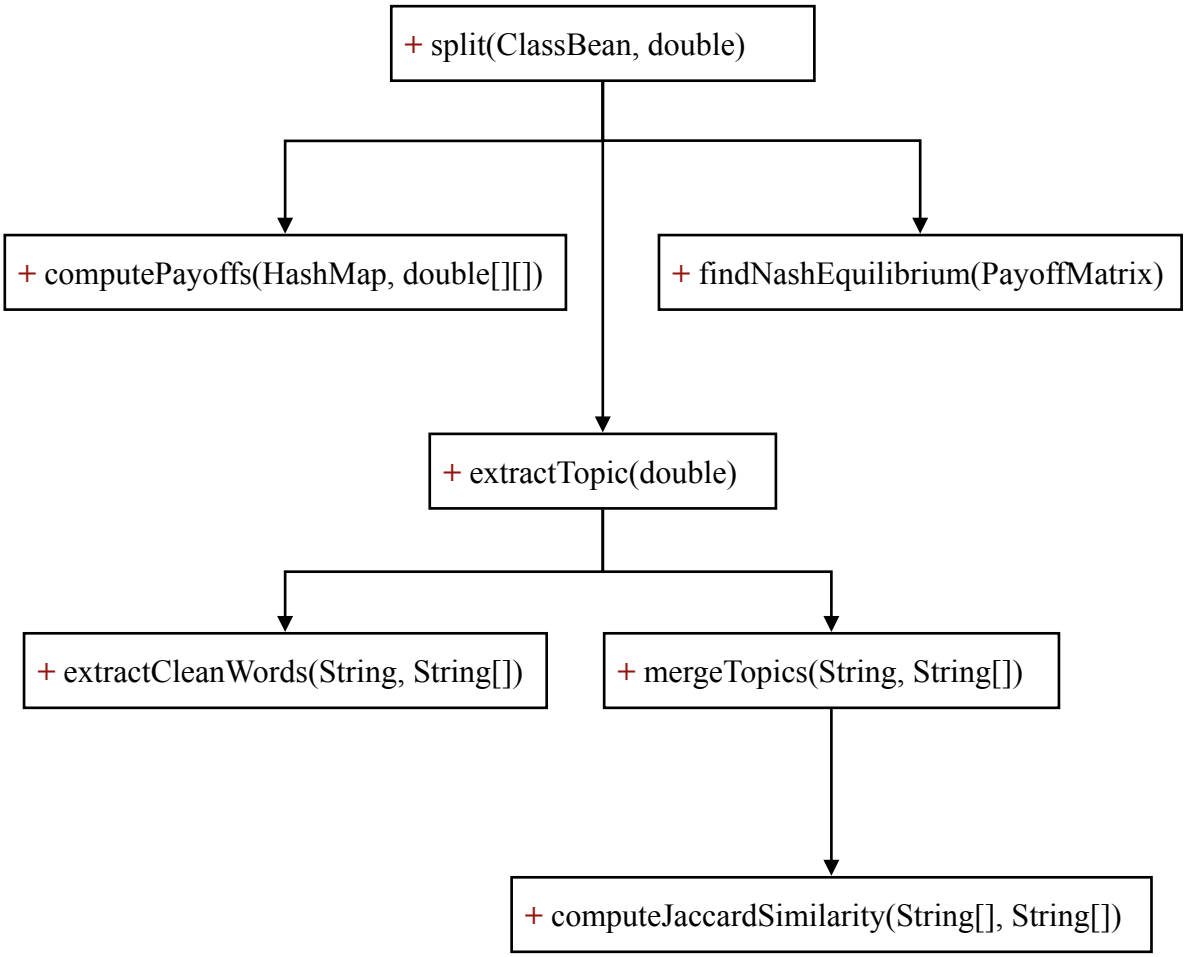


Figura 1: dependency graph

CR_02: Unit Testing

Le operazioni coinvolte nel test di unità sono:

ID	Classe	Metodo	Descrizione
IF_01	InputFinder.java	computeJaccardSimilarity	calcola la Jaccard Similarity tra due liste di stringhe
IF_02	InputFinder.java	extractCleanWords	applica una normalizzazione del testo di un metodo (eliminando caratteri speciali, parole contenute nella stop word list e tokenizzando i termini)
PM_01	PayoffMatrix.java	computePayoffs	calcola tutti i payoffs, in maniera ricorsiva, data una lista di possibili scelte
PM_02	PayoffMatrix.java	findNashEquilibrium	trova tutti i possibili equilibri di Nash data una matrice di payoffs

In particolare, per PM_01, sono state considerate le diverse tipologie di calcolo di payoffs corrispondenti a diverse combinazioni di scelte (input dei test case):

1. Un giocatore sceglie un metodo e lascia un insieme di metodi non vuoto agli altri giocatori.
2. Un giocatore sceglie un metodo e lascia un insieme di metodi vuoto agli altri giocatori (*i.e.* ogni altro giocatore fa la mossa nulla).
3. Un giocatore fa la mossa nulla.
4. Tutti i giocatori fanno la mossa nulla (combinazione non accettabile).

Questo garantisce un elevato livello di Branch Coverage.

I test case prodotti sono i seguenti:

ID	UT_IF_01
Nome metodo di test	canComputeJaccardSimilarity
Precondizioni	*InputFinder* instanziato
Input	set1 = {“utente", "nome", "password", "data", "luogo", "indirizzo", “azienda”} set2 = {“azienda", "nome", "titolo", "licenza", "fondazione", "luogo", "indirizzo", “data"}
Oracolo	0.5

ID	UT_IF_02
Nome metodo di test	canExtractCleanWords
Precondizioni	*InputFinder* instanziato
Input	testo = “public void creaUtente(String nome, String il@Cognome, integer eta){this.nome- := nome; sa}” set2 = {“public", "void", "string", "integer", "this"}
Oracolo	{“crea", "utente", "nome", "cognome", "eta", "nome", “nome"}

ID	UT_PM_01_1
Nome metodo di test	canComputeTheRightNumberOfPayoffs
Precondizioni	

Input	MethodByMethodMatrix = {
	{1.00, 0.70, 0.21, 0.02, 0.10, 0.00},
	{0.70, 1.00, 0.30, 0.06, 0.01, 0.03},
	{0.21, 0.30, 1.00, 0.50, 0.40, 0.22},
	{0.02, 0.06, 0.50, 1.00, 0.60, 0.30},
	{0.10, 0.01, 0.40, 0.60, 1.00, 0.80},
	{0.00, 0.03, 0.22, 0.30, 0.80, 1.00}
	}
	
	RemainingMethods = {0, 2, 5}
	PlayerChoices = {{1}, {4}, {3}}

Oracolo	Size = 33

ID	UT_PM_01_2
Nome metodo di test	canComputeTheRightPayoffs
Precondizioni	

Input	MethodByMethodMatrix = {
	{1.00, 0.70, 0.21, 0.02, 0.10, 0.00},
	{0.70, 1.00, 0.30, 0.06, 0.01, 0.03},
	{0.21, 0.30, 1.00, 0.50, 0.40, 0.22},
	{0.02, 0.06, 0.50, 1.00, 0.60, 0.30},
	{0.10, 0.01, 0.40, 0.60, 1.00, 0.80},
	{0.00, 0.03, 0.22, 0.30, 0.80, 1.00}
	}
	
	RemainingMethods = {0, 2, 5}
	PlayerChoices = {{1}, {4}, {3}}
	
	comb1 = {-1, -1, 2}
	comb2 = {2, 0, 5}
	comb3 = {2, -1, 0}
	comb4 = {-1, -1, -1}

Oracolo	expected1 = {0.2, 0.1, 0.1} expected2 = {-0.065, -0.5, 0.04} expected3 = {-0.4, 0.25, -0.48} expected4 = null

ID	UT_PM_02
Nome metodo di test	canFindNashEquilibrium
Precondizioni	

Input	PayoffMatrix = {
	(-1.00, -1.00), (-0.49, -0.22), (-0.70, -0.30), (-0.70, -0.80), (-0.20, 0.00),
	(0.49, 0.22), (-1.00, -1.00), (-0.21, -0.08), (-0.21, -0.58), (0.29, 0.22),
	(0.70, 0.30), (0.21, 0.08), (-1.00, -1.00), (0.00, -0.50), (0.50, 0.30),
	(0.70, 0.80), (0.21, 0.58), (0.00, 0.50), (-1.00, -1.00), (0.50, 0.80),
	(0.70, 0.50), (0.21, 0.28), (0.00, 0.20), (0.00, -0.30), (-1.00, -1.00),
	}

Oracolo	{(0.70, 0.80)}

CR_02: Integration Testing

Una volta testati i metodi terminali sono stati integrati con il metodo chiamante e si è testata la coppia. In seguito si è ridisegnato il grafo delle dipendenze sostituendo i due moduli integrati con un unico modulo e si è ripetuto iterativamente il procedimento fino ad ottenere un grafo con un unico nodo.

I test case prodotti in questa fase sono i seguenti:

ID	IT_IF_01
Nome metodo di test	canMergeTopics
Precondizioni	- *InputFinder* instanziato - il test UT_IF_01 passa

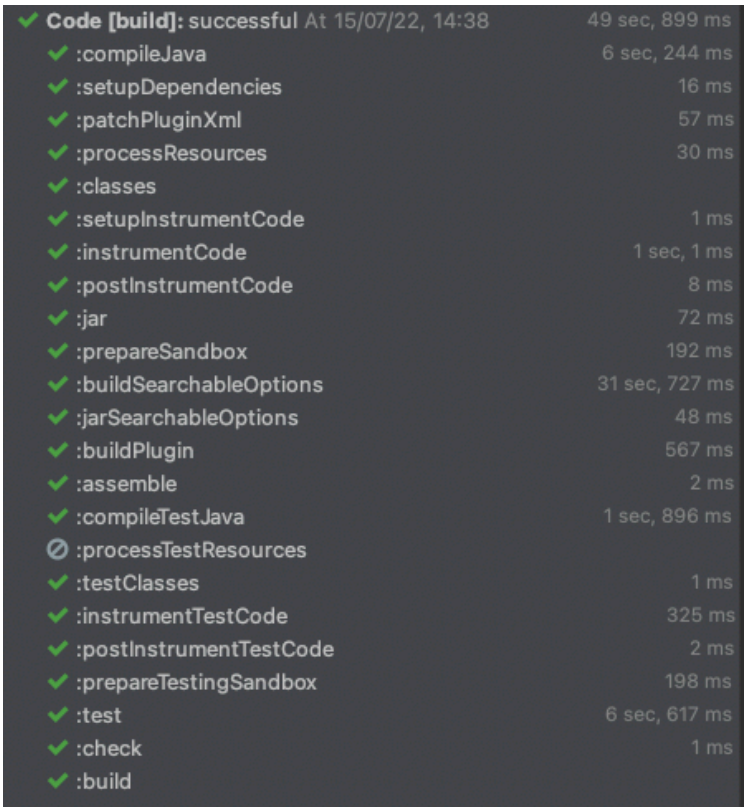
Input	set1 = {
	{“utente”, "nome", "password", "data", "luogo", "indirizzo", “azienda”},
	
	{“azienda", "nome", "titolo", "licenza", "fondazione", "luogo", "indirizzo", “data”},
	
	{“azienda", "nome", "via", “pIVA”},
	
	{“azienda", “nome", "fondazione", "utente", "licenza", “pIVA”, “data”}
	}

Oracolo	{
	{“password”, "titolo", "luogo", "indirizzo", "azienda", "nome", “fondazione”, “utente”, “licenza”, “pIVA”, “data”},
	
	{“azienda", "nome", "via", “pIVA”}
	}

ID	IT_IF_01
Nome metodo di test	canApplyLDA
Precondizioni	- *InputFinder* instanziato - il test UT_IF_01 passa - il test UT_IF_02 passa - il test IT_IF_01 passa
Input	ClassBean smelly Threshold = 1
Oracolo	Size = 5

Report di Esecuzione

Di seguito il report di esecuzione dei nuovi casi di test introdotti in seguito a CR_01 e CR_02.



GameTheorySplitClassesTest: 7 total, 7 passed 2.34 s

Collapse | Expand

it.unisa.casper.refactor.GameTheorySplitClassesTest		2.34 s
canFindNashEquilibrium	passed	126 ms
canComputeTheRightNumberOfPayoffs	passed	6 ms
canComputeJaccardSimilarity	passed	3 ms
canComputeTheRightPayoffs	passed	18 ms
canExtractCleanWords	passed	27 ms
canApplyLDA	passed	2.15 s
canMergeTopics	passed	3 ms

Generated by IntelliJ IDEA on 18/07/22, 11:43

Regression testing

Per il test di regressione non è stata necessaria un'ottimizzazione poiché, visto il numero di casi di test e le tempistiche di esecuzione, è stato utilizzato l'approccio Test All rieseguendo tutti i casi di test. Di seguito il report di esecuzione del test di regressione.

Code [test]: 90 total, 90 passed 5.12 s

Collapse | Expand

it.unisa.casper.analysis.code_smell_detection.blob.StructuralBlobStrategyTest		10 ms
isSmellyTrueControl	passed	3 ms
isSmellyMinThreshold	passed	2 ms
isSmellyFalse	passed	2 ms
isSmellyNearThreshold	passed	2 ms
isSmellyTrue	passed	1 ms
it.unisa.casper.analysis.code_smell_detection.blob.TextualBlobStrategyTest		113 ms
isSmellyMinThreshold	passed	39 ms
isSmellyFalse	passed	1 ms
isSmellyNearThreshold	passed	35 ms
isSmellyTrue	passed	38 ms
it.unisa.casper.analysis.code_smell_detection.feature_envy.StructuralFeatureEnvyStrategyTest		0 ms
isSmellyMinThreshold	passed	0 ms
isSmellyFalse	passed	0 ms
isSmellyNearThreshold	passed	0 ms
isSmellyTrue	passed	0 ms
it.unisa.casper.analysis.code_smell_detection.feature_envy.TextualFeatureEnvyStrategyTest		161 ms
isSmellyMinThreshold	passed	7 ms
isSmellyFalse	passed	0 ms
isSmellyNearThreshold	passed	5 ms
isSmellyTrue	passed	149 ms
it.unisa.casper.analysis.code_smell_detection.misplaced_class.StructuralMisplacedClassStrategyTest		1 ms
isSmellyMinThreshold	passed	1 ms
isSmellyFalse	passed	0 ms
isSmellyNearThreshold	passed	0 ms
isSmellyTrue	passed	0 ms
it.unisa.casper.analysis.code_smell_detection.misplaced_class.TextualMisplacedClassStrategyTest		27 ms
isSmellyMinThreshold	passed	6 ms
isSmellyFalse	passed	6 ms
isSmellyNearThreshold	passed	8 ms
isSmellyTrue	passed	7 ms

it.unisa.casper.analysis.code_smell_detection.promiscuous_package.StructuralPromiscuousPackageStrategyTest		0 ms
isSmellyMinThreshold	passed	0 ms
isSmellyFalse	passed	0 ms
isSmellyNearThreshold	passed	0 ms
isSmellyTrue	passed	0 ms
it.unisa.casper.analysis.code_smell_detection.promiscuous_package.TextualPromiscuousPackageStrategyTest		58 ms
isSmellyMinThreshold	passed	16 ms
isSmellyFalse	passed	1 ms
isSmellyNearThreshold	passed	18 ms
isSmellyTrue	passed	23 ms
it.unisa.casper.refactor.GameTheorySplitClassesTest		1.89 s
canFindNashEquilibrium	passed	2 ms
canComputeTheRightNumberOfPayoffs	passed	3 ms
canComputeJaccardSimilarity	passed	1 ms
canComputeTheRightPayoffs	passed	10 ms
canExtractCleanWords	passed	5 ms
canApplyLDA	passed	1.87 s
canMergeTopics	passed	4 ms
it.unisa.casper.refactor.SplitClassTest		26 ms
splitFalse	passed	17 ms
splitTrue	passed	9 ms
it.unisa.casper.refactor.SplitPackagesTest		54 ms
splitFalse	passed	1 ms
splitTrue	passed	53 ms

Generated by IntelliJ IDEA on 18/07/22, 11:38