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

DARTS_2.0

Test Plan

Sommario

Versione	***Modifiche***
0.1	Obiettivi del testing
0.2	Approccio al testing
0.3	Modifica Unit Test Plan
0.4	Generazione dei test frame
1.0	Testing di sistema
1.1	Revisione Finale

Autori	***Matricola***
Gerardo Iuliano	0522501329
Alessandro Bacco	0522501104
Tiziano La Monica	0522501258

Obiettivi del Testing

Come anticipato, 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 DARTS è un plugin per l'ambiente di sviluppo IntelliJ IDEA e in quanto tale questo vuol dire che in fase di identificazione dei test smell, sia nel refactoring, vengono utilizzati strumenti forniti dalla stessa piattaforma che sfruttano l'ambiente di esecuzione; ambiente che non può essere facilmente emulato in uno scenario di testing automatizzato. Questo però non ci impedisce del tutto di effettuare test di regressione. Infatti, visto la mancata possibilità di scrivere casi di test per il plugin, è stato concepito in precedenza un "test pilota", ovvero un test con delle istanze di test smell create a mano e che quindi potessero simulare l'oracolo. Lo scopo era far eseguire DARTS su questo progetto pilota e verificare che il plugin riuscisse a trovare ogni istanza di test smell inserita manualmente. Dopo il testing di regressione iniziale, che ci assicura che il plugin funzioni correttamente come descritto dalle documentazioni precedenti, si inizia a sviluppare le CR descritte. Dopo ogni task portato a termine, si passa al testing della CR e in caso di fault, si aggiusta la classe di produzione fino a quando il risultato del testing sia uguale all'oracolo. Dopo il testing della CR bisogna necessariamente eseguire di nuovo il testing di regressione, per avere la certezza che la nuova modifica effettuata non abbia intaccato il funzionamento di tutto il plugin.

Approccio

Si è deciso di seguire un approccio black-box basato sulle funzionalità di DARTS. Avendo aggiunto nuove funzionalità al plugin, sono state testate le seguenti:

- Il sistema deve consentire all'utente di individuare lo smell Magic Number nei test case.
- Il sistema deve consentire all'utente di individuare lo smell Conditional Test Logic nei test case.
- Il sistema deve consentire all'utente di individuare lo smell Exception Handling nei test case.
- Il sistema deve consentire all'utente di individuare lo smell Constructor Initialization nei test case.

Tali funzionalità, per essere testate, richiedono in input oggetti della classe PSIClass che rappresentano le classi di test del progetto sul quale il plugin viene lanciato. Si è quindi deciso di applicare Category Partition e di usare come categoria il numero di istanze di smells presenti in un progetto per il testing di sistema mentre per il testing di unità e di integrazione la categoria sarà la presenza o meno del progetto.

Data la natura del testing su un plugin, per simulare come input un progetto sarà utilizzata la classe LightJavaCodeInsightFixtureTestCase. Il testing non prende in input un progetto vero e proprio ma attraverso l'uso di LightJavaCodeInsightFixtureTestCase si simula la presenza di un progetto nel quale le classi di produzione e le classi di testing sono impostate dal tester. Il progetto viene creato a partire da delle classi target scelte dal tester. Di conseguenza come input del parametro Project vengono date una o più classi partendo dalle quali viene creato un progetto fittizio da LightJavaCodeInsightFixtureTestCase.

Esempio:
Progetto = "MagicNumberPresent.java"
Data la classe "MagicNumberPresent.java", LightJavaCodeInsightFixtureTestCase crea un'istanza di project composta dalla classe "MagicNumberPresent.java". Di conseguenza al parametro Progetto viene assegnata l'istanza di un project composto dalla classe indicata. Nei test case saranno quindi indicate le classi che compongono l'istanza di project.

Infine è stato effettuato un ulteriore testing white-box per aumentare la coverage di ogni detector.

Si è scelto il criterio di copertura branch coverage basato sui Control Flow Graph di ogni detector. Per massimizzare la coverage saranno coperti i branch di ogni CFG non coperti dal testing black box. Per ulteriori dettagli consultare il paragrafo relativo al testing white box.

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.

Unit Test Plan

Nel test di unità si sono andate a testare principalmente le classi coinvolte nel processo di detection. Essendo l'obiettivo del testing verificare la presenza di faults all'interno del codice introdotto, è stato necessario testare tutte le singole classi che partecipano alla funzionalità sotto testing. Testare in modo isolato ogni singola componente permette di escludere eventuali malfunzionamenti derivanti da classi di supporto, concentrando così l'attenzione sulle componenti nuove. In particolare, le classi di supporto al detector sono:

Test item

ID	***Class***	***Caratteristica da testare***
CU_Unit	ConverterUtilities.java	Dato in input un progetto, si occupa di estrarre tutte le classi dai packages?
TSU_Unit	TestSmellUtilities.java	Dato in input le classi di un progetto, estrae tutte le classi di test?

Criteri pass/fail

Fail	Il test non ha trovato nessun fault.
Pass	Il test ha trovato un fault.

CU_Unit	
Parametri	• Project
Categorie	• C_1: progetto presente
Scelte	**C_1** • *SP_1*: Progetto not null: • *SP_2*: Progetto null
Vincoli	Nessun vincolo

Codice	***Test Frame***	***Oracolo***
testConverterUtilitiesNotNull	*SP_1*	Progetto individuato e classi estratte.
testConverterUtilitiesNull	*SP_2*	Progetto non individuato e nessuna classe estratta.

TSU_Unit	
Parametri	• PSIClass
Categorie	• C_1: classi presenti
Scelte	**C_1** • *SC_1*: Classi not null: • *SC_2*: Classi null
Vincoli	Nessun vincolo

Codice	***Test Frame***	***Oracolo***
testSmellUtilitiesNotNull	*SC_1*	Le classi sono presenti e vengono estratte solo le classi di test
testSmellUtilitiesNull	*SC_2*	Le classi non sono presenti

Integration Test Plan

Nel test di integrazione sono state testate insieme le componenti di supporto alla detection.

Test item

ID	***Classes***	***Caratteristica da testare***
CU_TSU_Integration	ConverterUtilities.java TestSmellUtilities.java	Dato in input un progetto, ConverterUtilities.java si occupa di estrarre tutte le classi dai packages. Le classi ottenute sono date in input a TestSmellUtilities.java che estrae esclusivamente tutte le classi di test.

Criteri pass/fail

Fail	Il test non ha trovato nessun fault.
Pass	Il test ha trovato un fault.

Parametri	• Project
Categorie	• C_1: progetto presente
Scelte	**C_1** • *SP_1*: Progetto not null • *SP_2*: Progetto null

Parametri	• PSIClass
Categorie	• C_1: classi presenti
Scelte	**C_1** • *SC_1*: Classi not null • *SC_2*: Classi null

Vincoli	• *SC_1* [*if* Progetto not null]

Codice	***Test Frame***	***Oracolo***
testIntegrationUtilitiesNotNull	*SP_1, SC_1*	Le classi di test vengono estratte
testIntegrationUtilities_ClassNull	*SP_1, SC_2*	Nessuna classe di test estratta
testIntegrationUtilitiesNull	*SP_2, SC_2*	Nessuna classe di test estratta

System Test Plan

Nel test di Sistema sono state unite tutte le componenti che partecipano alla detection di uno smell. Tali componenti rappresentano l'intero flusso di esecuzione del plugin per la detection degli smells.

Test item

ID	***Class***	***Caratteristiche da testare***
CTL_System	ConverterUtilities.java TestSmellUtilities.java ConditionalTestLogicStructural.java	Test di sistema sulla funzionalità di detection dello smell Conditionl Test Logic
EH_System	ConverterUtilities.java TestSmellUtilities.java ExceptionHandlingStructural.java	Test di sistema sulla funzionalità di detection dello smell Exception Handling
CI_System	ConverterUtilities.java TestSmellUtilities.java ConstructorInitializationStructural.java	Test di sistema sulla funzionalità di detection dello smell Constructor Initialization
MN_System	ConverterUtilities.java TestSmellUtilities.java MagicNumberStructural.java	Test di sistema sulla funzionalità di detection dello smell Magic Number
DA_System	ConverterUtilities.java TestSmellUtilities.java DuplicateAssertStructural.java	Test di sistema sulla funzionalità di detection dello smell Duplicate Assert
IT_System	ConverterUtilities.java TestSmellUtilities.java IgnoredTestStructural.java	Test di sistema sulla funzionalità di detection dello smell Ignored Test

Criteri pass/fail

Fail	Il test non ha trovato nessun fault.
Pass	Il test ha trovato un fault.

Il seguente studio mostra i parametri in input necessari al testing di sistema del detector ConditionalTestLogic (CTL_System) e ExceptionHandling (EH_System). Per testare la funzionalità di detection è necessario prendere in input un *Project* e una *Threshold* inserita dall'utente. Essendo che tali detector prevedono gli stessi parametri di input, lo studio seguente è condiviso per i detector sopra citati.

Generati i Test Frame invece, la definizione dei test case sarà fatta per ogni detector.

Parametri	• Project p
Categorie	• C_1: istanze di smell nel project
Scelte	**C_1** • *PS_1*: Progetto con 0 istanze di smell • *PS_2*: Progetto con 1 istanza di smell • *PS_3:* Progetto con 2 o più istanze di smell

Parametri	• Threshold x
Categorie	• C_1: valore
Scelte	**C_1** • Valore in-range: ▪ *TS_1*: 0 ▪ *TS_2*: 1-4 ▪ *TS_3*: 5 • Valore out-of-range: ▪ *TS_4*: -1 ▪ TS_5: 6

Vincoli	• *TS_1, TS_2, TS_3, TS_4, TS_5* [*if* progetto presenta classi di test] • *TS_1, TS_2, TS_3, TS_4, TS_5* [*if* progetto not null]

Test Frame CTL_System

Codice	***Test Frame***	***Oracolo***
TC_Sys_01	*PS_1, TS_1*	Il detector non individua nessuno smell
TC_Sys_02	*PS_1, TS_2*	Il detector non individua nessuno smell
TC_Sys_03	*PS_1, TS_3*	Il detector non individua nessuno smell
TC_Sys_04	*PS_1, TS_4*	Il plugin segnala errore dell'input e non effettua la detection
TC_Sys_05	*PS_1, TS_5*	Il plugin segnala errore dell'input e non effettua la detection
TC_Sys_06	*PS_2, TS_1*	Il detector individua gli smell che rispettano la threshold
TC_Sys_07	*PS_2, TS_2*	Il detector individua gli smell che rispettano la threshold
TC_Sys_08	*PS_2, TS_3*	Il detector individua gli smell che rispettano la threshold
TC_Sys_09	*PS_2, TS_4*	Il plugin segnala errore dell'input e non effettua la detection
TC_Sys_10	*PS_2, TS_5*	Il plugin segnala errore dell'input e non effettua la detection

TC_Sys_11	*PS_3, TS_1*	Il detector individua gli smell che rispettano la threshold
TC_Sys_12	*PS_3, TS_2*	Il detector individua gli smell che rispettano la threshold
TC_Sys_13	*PS_3, TS_3*	Il detector individua gli smell che rispettano la threshold
TC_Sys_14	*PS_3, TS_4*	Il plugin segnala errore dell'input e non effettua la detection
TC_Sys_15	*PS_3, TS_5*	Il plugin segnala errore dell'input e non effettua la detection

Test Frame EH_System

Codice	***Test Frame***	***Oracolo***
TC_Sys_16	*PS_1, TS_1*	Il detector non individua nessuno smell
TC_Sys_17	*PS_1, TS_2*	Il detector non individua nessuno smell
TC_Sys_18	*PS_1, TS_3*	Il detector non individua nessuno smell
TC_Sys_19	*PS_1, TS_4*	Il plugin segnala errore dell'input e non effettua la detection
TC_Sys_20	*PS_1, TS_5*	Il plugin segnala errore dell'input e non effettua la detection
TC_Sys_21	*PS_2, TS_1*	Il detector individua gli smell che rispettano la threshold
TC_Sys_22	*PS_2, TS_2*	Il detector individua gli smell che rispettano la threshold
TC_Sys_23	*PS_2, TS_3*	Il detector individua gli smell che rispettano la threshold
TC_Sys_24	*PS_2, TS_4*	Il plugin segnala errore dell'input e non effettua la detection
TC_Sys_25	*PS_2, TS_5*	Il plugin segnala errore dell'input e non effettua la detection
TC_Sys_26	*PS_3, TS_1*	Il detector individua gli smell che rispettano la threshold
TC_Sys_27	*PS_3, TS_2*	Il detector individua gli smell che rispettano la threshold
TC_Sys_28	*PS_3, TS_3*	Il detector individua gli smell che rispettano la threshold
TC_Sys_29	*PS_3, TS_4*	Il plugin segnala errore dell'input e non effettua la detection
TC_Sys_30	*PS_3, TS_5*	Il plugin segnala errore dell'input e non effettua la detection

Per quanto riguarda il test di sistema sui detector per gli smell Constructor Initialization, Magic Number, Duplicate Assert e Ignored Test, tali detector prendono in input solo l'istanza del progetto e non necessitano di una Threshold.

CI_System, MN_System, DA_System e IT_System condividono quindi la seguente tabella:

Parametri	• Project p
Categorie	• C_1: istanze di smell nel project
Scelte	**C_1** • *PS_1*: Progetto con 0 istanze di smell • *PS_2*: Progetto con 1 istanza di smell • *PS_3:* Progetto con 2 o più istanze di smell

Vincoli	Nessun Vincolo

Test Frame CI_System

Codice	***Test Frame***	***Oracolo***
TC_Sys_31	*PS_1*	Il detector non individua nessuno smell
TC_Sys_32	*PS_2*	Il detector individua 1 smell
TC_Sys_33	*PS_3*	Il detector individua 2 o più smell

Test Frame MN_System

Codice	***Test Frame***	***Oracolo***
TC_Sys_34	*PS_1*	Il detector non individua nessuno smell
TC_Sys_35	*PS_2*	Il detector individua 1 smell
TC_Sys_36	*PS_3*	Il detector individua 2 o più smell

Test Frame DA_System

Codice	***Test Frame***	***Oracolo***
TC_Sys_37	*PS_1*	Il detector non individua nessuno smell
TC_Sys_38	*PS_2*	Il detector individua 1 smell
TC_Sys_39	*PS_3*	Il detector individua 2 o più smell

Test Frame IT_System

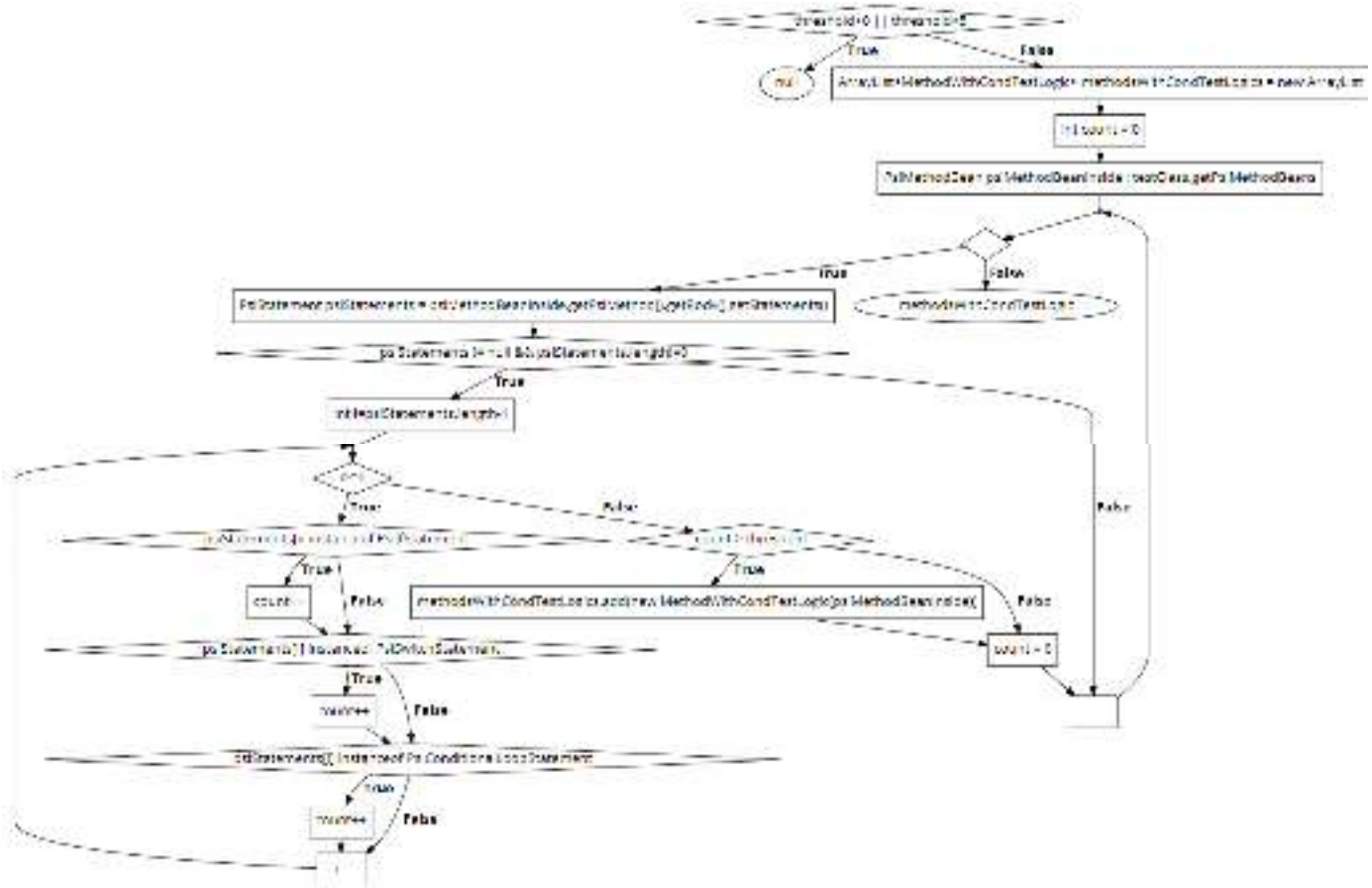
Codice	***Test Frame***	***Oracolo***
TC_Sys_40	*PS_1*	Il detector non individua nessuno smell
TC_Sys_41	*PS_2*	Il detector individua 1 smell
TC_Sys_42	*PS_3*	Il detector individua 2 o più smell

Control Flow Graph

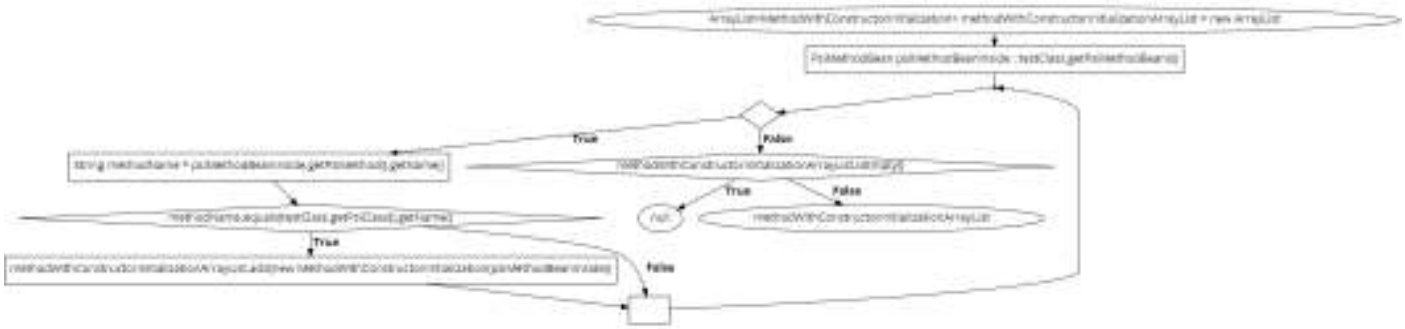
Di seguito sono riportati i CFG dei detector che saranno utilizzati per identificare quali condizioni soddisfare al fine di incrementare la coverage. I seguenti CFG saranno confrontati con il report di jacoco. Per ogni branch non evidenziato nel report di jacoco sarà creato un test case che copre quel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

I seguenti CFG sono stati generati dal tool Code2Flow.

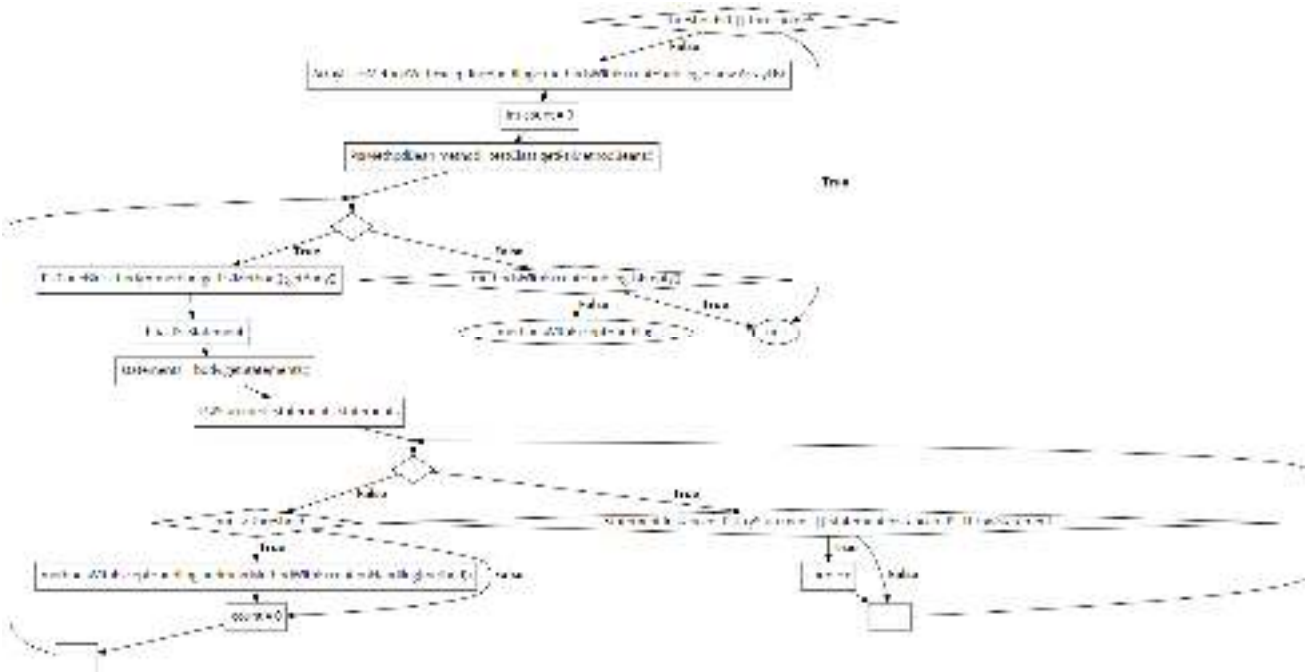
- *ConditionalTestLogic*



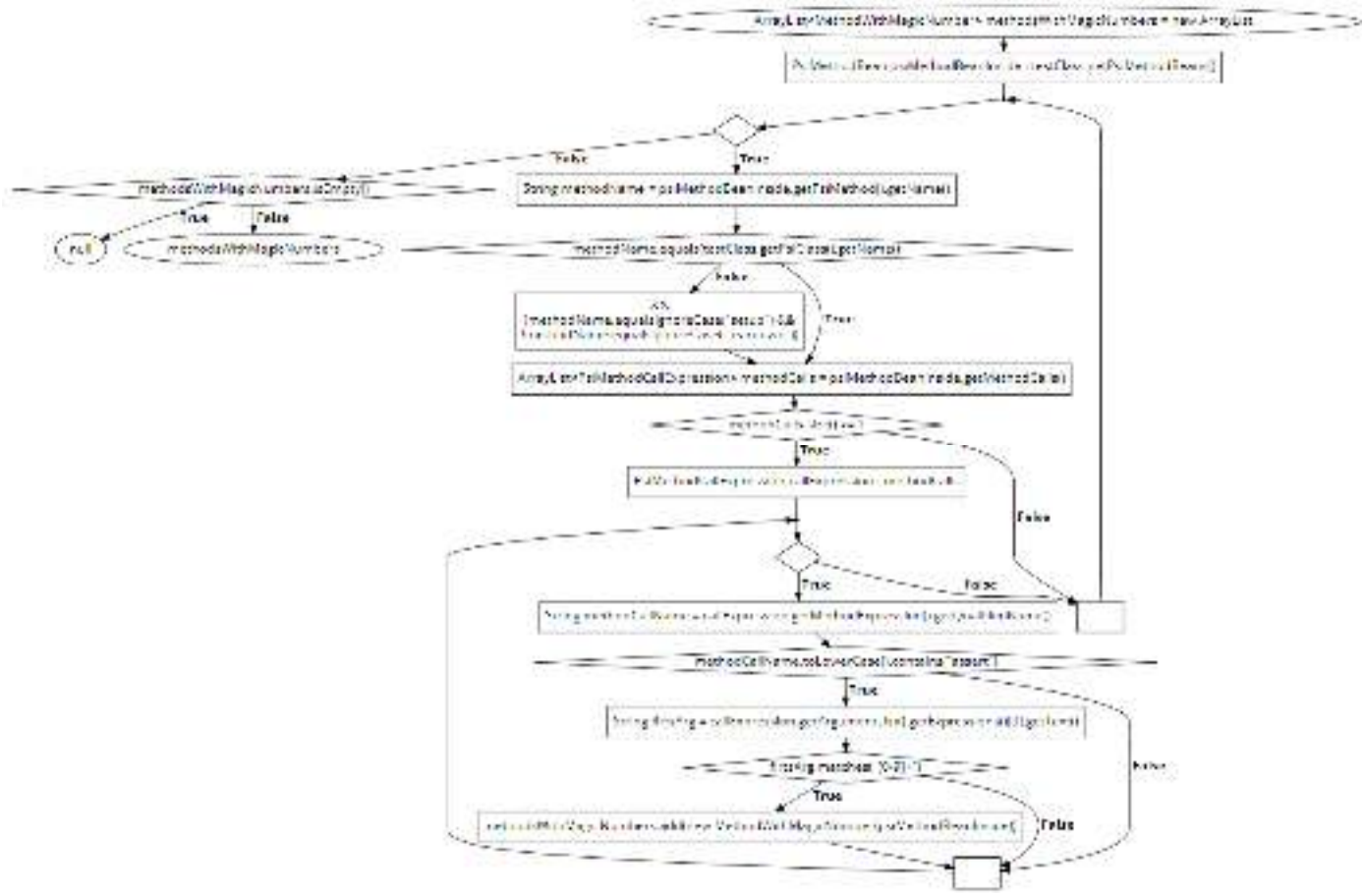
- *ConstructorInitialization*



- *ExceptionHandling*



- *MagicNumber*



Test di Regressione

In assenza di un'automazione dell'operazione di testing all'interno del tool, è stato necessario effettuare il test manualmente analogamente a quanto riportato nella documentazione della versione precedente di DARTS.

Quindi, è stato utilizzato il progetto fantoccio TestProjectForDarts, il quale presenta classi di test per ogni tipologia di test smell.

Il progetto TestProjectForDARTS è un progetto fantoccio utilizzato nella precedente versione del plugin per effettuare il testing di sistema.

Grazie all'esecuzione del plugin sul progetto TestProjectForDARTS è stato appurato che tra le funzionalità del sistema, la parte relativa al refactoring non funziona correttamente. Il plugin presenta la possibilità di effettuare il refactoring del codice affetto da smell tramite la pressione di un button presente sulla GUI. Alla pressione del button il plugin non effettua nessun refactoring rimanendo il codice affetto da smell inalterato. L'unica funzionalità relativa al refactoring è il suggerimento che il plugin propone per eliminare lo smell. Tale suggerimento viene mostrato nella GUI in corrispondenza del codice affetto da smell.

In seguito a questo risultato, abbiamo creato dei test case relativi alle funzionalità del plugin che non hanno mostrato problemi, in quanto il testing effettuato nella versione precedente non è sufficiente a darci una risposta affidabile. Sono stati quindi testati i detector del plugin. Tale testing ha mostrato l'effettivo funzionamento delle restanti funzionalità.

Il testing di regressione è stato lanciato prima di apportare le modifiche richieste dalle CR, al termine dell'implementazione di ogni CR e alla fine di tutte le modifiche apportate al sistema.

Nessuna funzionalità del sistema è stata intaccata dalle CR.

Di seguito è riportata la pianificazione dei test sulle funzionalità di detection del plugin originale. Il category partition applicato sul parametro project non è stato applicato in maniera esaustiva in quanto questo testing è stato fatto solo per confermare il funzionamento dei detector.

Test item

ID	***Classes***	***Caratteristica da testare***
ET_System	ConverterUtilities.java TestSmellUtilities.java EagerTestStructural.java	Il detector è in grado di identificare lo smell?
GF_System	ConverterUtilities.java TestSmellUtilities.java GeneralFixtureStructural.java	Il detector è in grado di identificare lo smell?
LOC_System	ConverterUtilities.java TestSmellUtilities.java LackOfCohesionOfTextSmellTextual.java	Il detector è in grado di identificare lo smell?

Criteri pass/fail

Fail	Il test non ha trovato nessun fault.
Pass	Il test ha trovato un fault.

Parametri	• Project p
Categorie	• C_1: istanze di smell nel project
Scelte	**C_1** • *PS_1*: Progetto contiene 1 o più istanze di smell • *PS_2*: Progetto non contiene istanze di smell

Vincoli	Nessun Vincolo

Test Frame ET_System

Codice	***Test Frame***	***Oracolo***
TC_Reg_Sys_1	*PS_1*	Il detector individua uno o più smell
TC_Reg_Sys_2	*PS_2*	Il detector non individua nessuno smell

Test Frame GF_System

Codice	***Test Frame***	***Oracolo***
TC_Reg_Sys_3	*PS_1*	Il detector individua uno o più smell
TC_Reg_Sys_4	*PS_2*	Il detector non individua nessuno smell

Test Frame LOC_System

Codice	***Test Frame***	***Oracolo***
TC_Reg_Sys_5	*PS_1*	Il detector individua uno o più smell
TC_Reg_Sys_6	*PS_2*	Il detector non individua nessuno smell