

UNIVERSITÀ DEGLI STUDI DI SALERNO

DIPARTIMENTO DI INFORMATICA



CORSO DI LAUREA MAGISTRALE IN INFORMATICA
INGEGNERIA, GESTIONE ED EVOLUZIONE DEL SOFTWARE

Analisi di Manutenzione

Docente del corso:

Andrea De Lucia

Studentesse:

Mariarosaria Esposito
ME - 0522501095

Francesca Perillo
FP - 0522501096

ANNO ACCADEMICO 2021/2022

Revision History

Data	Versione	Descrizione	Autori
07/04/2022	1.0	Studio del problema e inizio prima stesura	All Team
20/06/2022	2.0	Passaggio a Gradle e panoramica dell'integrazione.	All Team
23/06/2022	3.0	Studio e stesura dell'Impact Analysis CR_1	All Team
24/06/2022	4.0	Stesura dei documenti di testing CR_1 .	All Team
04/07/2022	5.0	Studio e inizio stesura dell'Impact Analysis CR_2 .	All Team
11/07/2022	6.0	Stesura dei documenti di testing CR_2 .	All Team
01/08/2022	7.0	Studio e inizio stesura dell'Impact Analysis CR_3 .	All Team
19/08/2022	8.0	Stesura dei documenti di testing CR_3 .	All Team

Indice

1	Panoramica del sistema attuale	3
1.1	Architettura di DARTS	3
2	Nuove funzionalità per DARTS	5
2.1	Passaggio ad un progetto Gradle	5
3	Panoramica dell'attività di integrazione	7
4	Prima Change Request $[CR_1]$	9
4.1	Impact Analysis	9
4.1.1	Starting Impact Set - SIS	9
4.2	Testing	10
4.3	Architettura del sistema	10
5	Seconda Change Request $[CR_2]$	13
5.1	Impact Analysis	13
5.1.1	Starting Impact Set - SIS	13
5.2	Testing	14
5.3	Architettura del sistema	14
6	Terza Change Request $[CR_3]$	17
6.1	Impact Analysis	17
6.1.1	Starting Impact Set - SIS	17
6.2	Testing	19
6.3	Architettura del sistema	19
7	Abbreviazioni	21

Capitolo 1

Panoramica del sistema attuale

In questo capitolo verrà introdotto il sistema DARTS, la cui documentazione è disponibile nella repository pubblica GitHub [1]. La documentazione di tale sistema risulta essere molto chiara ed esaustiva e correttamente aggiornata, per cui non è stata necessaria una previa attività di *Reverse Engineering*. Analizzando la documentazione, riusciamo quindi bene a comprendere quali sono le caratteristiche del sistema. DARTS (**D**etection and **R**efactoring of **T**est **S**melles) [2] è un plug-in IntelliJ che implementa un meccanismo di rilevamento di istanze di alcuni tipi di Test Smell. Tra questi ritroviamo *General Fixture* (GF), *Eager Test* (ET) e *Lack of Cohesion of Test Methods* (LCTM).

1.1 Architettura di DARTS

Volendo analizzare l'architettura di questo tool, vediamo che si compone di due livelli principali: *Interface* e *Application Layers*; ognuno di questi si compone a sua volta di vari sottosistemi (vedi Figura 1.1).

🔗 **Interface Layer.** Gestisce le interazioni degli utenti con il sistema. All'interno di questo livello, vi sono altri due sottosistemi:

- il pacchetto *extension* implementa la funzionalità che consente a DARTS di restare in ascolto delle attività svolte da uno sviluppatore e interagire con quest'ultimo quando esegue un commit;
- il pacchetto *WindowsConstruction* è responsabile della definizione dell'interfaccia grafica utente e delle componenti UI che l'utente usa per interagire con il sistema.

🔗 **Application Layer.** Contiene l'intera logica del plug-in. All'interno di questo livello, vi sono altri due sottosistemi:

- il pacchetto *testSmellDetection* contiene le regole per il rilevamento dei Test Smell, una per ogni Smell considerato;
- il pacchetto *refactoring* riguarda le operazioni di refactoring implementate dal tool.

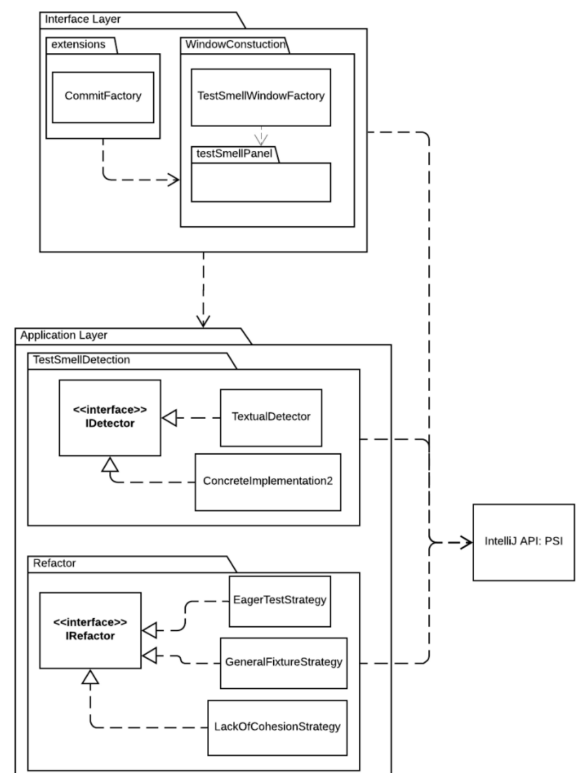


Figura 1.1: Architettura di DARTS

Capitolo 2

Nuove funzionalità per DARTS

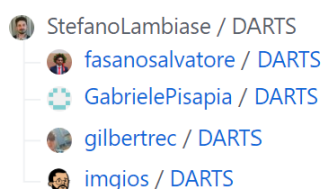


Figura 2.1: DARTS GitHub

Sul progetto principale *StefanoLambiasi/DARTS* sono state fatte un totale di quattro Forks (vedi Figura 2.1), ovvero: *fasanosalvatore/DARTS*, *GabrielePisapia/DARTS*, *gilbertrec/DARTS* e *imgios/DART*. Nello specifico:

- 🔗 ***fasanosalvatore/DARTS***: questa fork prevede il passaggio a un progetto Gradle e l'aggiunta di tre test smells al progetto padre, ovvero *Mystery Guest* (MG), *Hard Coded Test Data* (HCTD) e *Test Code Duplication* (TCD).
- 🔗 ***GabrielePisapia/DARTS***: non viene presentata alcuna nuova documentazione ma, analizzando il codice, vi sono alcuni contributi al progetto principale. Da una analisi più approfondita, il progetto sembra essere stato abbandonato, in quanto le implementazioni sono state solo iniziate, ma non concluse;
- 🔗 ***gilbertrec/DARTS***: questa fork prevede il passaggio a un progetto Gradle e l'aggiunta di un sottosistema in grado di condurre alcune statistiche di utilizzo del tool, con un resoconto di varie misure (ad esempio: il numero dei click, il numero degli smells identificati, il tempo di esecuzione ...) che sia esportabile. Inoltre, tali statistiche dovranno essere inviate ad un server remoto, denominato *DARTStats*.
- 🔗 ***imgios/DART***: in questo caso, è auspicabile che sia stata fatta una fork mirata a salvare lo stato attuale del progetto, in quanto non vi sono cambiamenti al progetto padre.

2.1 Passaggio ad un progetto Gradle

Considerando quindi esclusivamente le forks che hanno dato un contributo significativo alla versione originale del Tool IntelliJ (vedi Figura 2.2), da una prima analisi comparativa emerge il passaggio ad un progetto che, rispetto al progetto padre, segue un nuovo standard basato su *Gradle*. La domanda che ci poniamo è se effettivamente mantenere questa modifica abbia o meno senso. Gradle offre un supporto per la gestione delle dipendenze che aiuta il processo di build. L'utilizzo di Gradle migliora la flessibilità e le prestazioni. La decisione di utilizzare questo standard è stata presa al fine di avere un progetto che segua il nuovo standard IntelliJ per i propri plug-in, quindi riteniamo opportuno mantenere tale modifica per due motivi principali. In primo luogo, perché se così non fosse, il plug-in DARTS

non sarebbe più supportato dalle nuove edizioni degli IDE IntelliJ. In secondo luogo, mantenere questa modifica permetterà di poter sfruttare al meglio le novità introdotte dall'API della piattaforma IntelliJ.

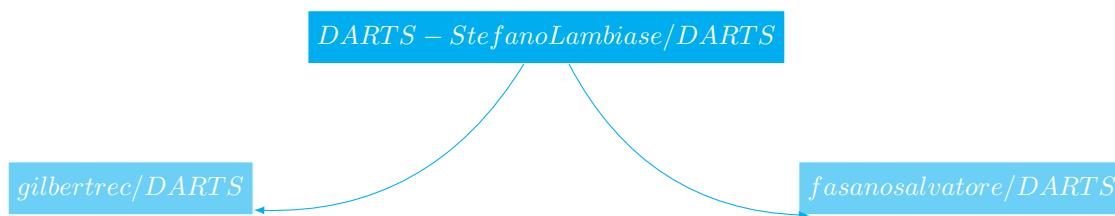


Figura 2.2: Contributi per DARTS

Capitolo 3

Panoramica dell'attività di integrazione

L'obiettivo cardine di questo paper è quello di documentare l'attività di integrazione e le nuove funzionalità per lo stesso, introdotte nei capitoli precedenti. La prima candidata a prendere parte al processo di integrazione sarà la fork *gilbertrec/DARTS*, alla quale seguirà la fork di *fasanosalvatore/DARTS*.

Ognuno dei processi di integrazione sarà una Change Request (CR). Lo svolgimento di queste ultime procederà in maniera sequenziale, il che significa che solo una volta completata l'integrazione della prima CR, si procederà con la seconda. Terminata l'integrazione ci saranno alcune implementazioni non allineate, come vedremo nei capitoli che seguono. A tal proposito, si avrà una terza CR che consiste nell'allineamento del sistema, dunque verranno apportate le modifiche necessarie al fine di ottenere un'ultima versione che risulti essere perfettamente funzionante e consta di tutte le funzionalità aggiuntive date dalle due forks.



In altre parole, nel corso del documento verranno integrate le modifiche effettuate dalle repository *gilbertrec/DARTS* e *fasanosalvatore/DARTS* nel seguente modo:

1. verranno integrate le funzionalità della repository *gilbertrec/DARTS* all'interno del progetto originale. Per effettuare il testing in questa fase verranno utilizzati i medesimi test utilizzati dal Team di Gilberto al fine di verificare che l'integrazione sia andata a buon fine. I test realizzati dal Team in questione sono basati sul lancio del sistema DARTS per verificare che il comportamento effettivo rispetti il comportamento atteso;
2. verranno integrate le funzionalità della repository *fasanosalvatore/DARTS* all'interno del progetto integrato al punto 1. Per effettuare il testing in questa fase verranno utilizzati sia i test del Team di Gilberto che i test del team di Salvatore. In questo modo avremo la certezza che le modifiche apportate al progetto dal Team di Gilberto non siano state compromesse dalle modifiche apportate dal Team di Salvatore. Verificheremo inoltre che le modifiche di quest'ultimo siano correttamente funzionanti. Mentre i test del Team di Gilberto sono basati sul lancio del sistema DARTS per verificare che il comportamento effettivo rispetti il comportamento atteso, il Team di Salvatore ha realizzato dei test sfruttando il potenziale di Junit.
3. verrà necessariamente prevista una fase di allineamento. In questa fase verranno eseguiti gli stessi test effettuati nei punti 1 e 2. Inoltre, verranno inseriti dei nuovi casi di test in modo da testare che l'allineamento sia andato a buon fine.

Capitolo 4

Prima Change Request $[CR_1]$



La prima Change Request prevede l'integrazione fra i due sistemi:  **StefanoLambiasi/DARTS**, ovvero il sistema originale di DARTS e  **gilbertrec/DARTS**, ovvero una delle fork analizzate nel Capitolo 2.

L'impatto delle modifiche verrà valutato utilizzando una scala consta di tre valori:

- *Forte*: il sistema subisce modifiche sostanziali o deve essere sostituito;
- *Medio*: il sistema subisce modifiche non banali, senza quindi cambiare radicalmente la sua struttura;
- *Debole*: il sistema subisce solo modifiche marginali.

4.1 Impact Analysis

Il codice risulta essere molto semplice e inoltre, essendo accompagnato anche da una documentazione, è stato semplice comprendere quali modifiche dovessero essere fatte al fine di inglobare all'interno del codice tutte le modifiche che consentono di effettuare questa prima CR; avremmo quindi che $|SIS| = |CIS| = |AIS|$. Andiamo prima di tutto a vedere quali sono le classi impattata nello *Starting Impact Set*, per poi effettuare il calcolo delle metriche tradizionali per valutare la bontà dell'Impact Analysis.

4.1.1 Starting Impact Set - SIS

Artefatto	Impatto	Descrizione
Nuovo Panel per visionare l'interfaccia delle statistiche	Medio	Per permettere alle statistiche di essere visionate su UI del tool bisogna implementare un Frame contenente i risultati delle statistiche.
main.java.commitWindowConstruction. commitWindowFactor	Debole	Per aggiungere il nuovo Panel per visionare le statistiche è necessario aggiungerlo in questa classe.

Alcune classi java all'interno del package <code>main.java.action</code> : 1. <code>StructuralDetectionAction.java</code> 2. <code>TextualDetectionAction.java</code>	Debole	Per il calcolo di alcune statistiche devono essere modificate le classi (1) e (2) in modo da calcolare le statistiche dall'avvio del plugin.
Alcune classi java all'interno del package <code>main.java.refactor.strategy</code> : 1. <code>EagerTestStrategy.java</code> 2. <code>GeneralFixtureStrategy.java</code> 3. <code>LackOfCohesonStrategy.java</code>	Debole	Modifiche di codice per permettere di calcolare le statistiche sugli smell ET, GF, LOC.

A questo punto è utile riportare le cardinalità degli insiemi impattati. Per quanto detto precedentemente, gli insiemi avranno tutti la stessa cardinalità. Abbiamo quindi che:

- $|SIS| = |CIS| = |AIS| = 7$;
- $|FPIS| = |CIS| - |AIS| = 0$;

Avendo la cardinalità di tutti gli insiemi, andiamo a calcolare le metriche tradizionali valutare la bontà dell'Impact Analysis: *Recall* e *Precision*. Al fine di calcolare il valore di entrambe, dobbiamo dapprima calcolare la cardinalità dell'insieme intersezione fra *CIS* e *AIS*.

- $|CIS \cap AIS| = 7$;
- **Recall** = $\frac{|CIS \cap AIS|}{|AIS|} = \frac{7}{7} = 1$, questo vuol dire che non c'è stato nessun componente individuato durante la modifica che non sia stato anche individuato nel CIS.
- **Precision** = $\frac{|CIS \cap AIS|}{|CIS|} = \frac{7}{7} = 1$, valore conforme al fatto che *FPIS* è vuoto, in quanto la sua cardinalità è pari a 0.

4.2 Testing

Il testing di questa prima Change Request è stilato al **Capitolo 1** del *Documento di Testing*.

4.3 Architettura del sistema

L'architettura del sistema dopo la prima fase di integrazione è cambiata, è possibile visionare tale cambiamento alla Figura 4.1

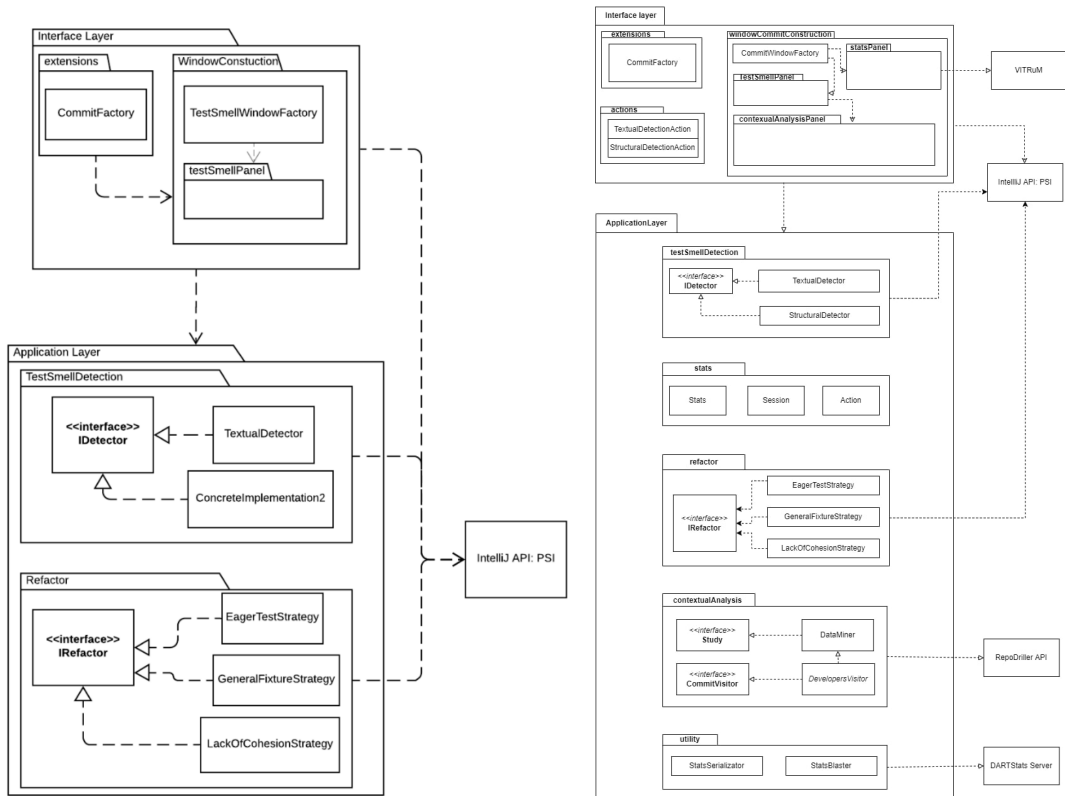


Figura 4.1: A sinistra l'architettura di DARTS prima della fase di integrazione, a destra la nuova architettura del sistema DARTS dopo la prima fase di integrazione.

Capitolo 5

Seconda Change Request $[CR_2]$



Ricordiamo che questa seconda Change Request prevede l'integrazione del nuovo progetto dato dall'integrazione della repository principale con quella di *gilbertrec/DARTS*, alla repository di *fasanosalvatore/DARTS*. Anche per quanto riguarda CR_2 , l'aspetto fondamentale è quello del passaggio ad un progetto che segue Gradle. Un fattore importante è quindi quello di assicurarsi che le due versioni di Gradle siano le medesime.

Come già detto in precedenza, la repository di *fasanosalvatore/DARTS* aggiunge al tool DARTS tre test smell. Quello che è stato fatto quindi è stato aggiungere alla precedente integrazione tutte le implementazioni necessarie al fine di *preservare il comportamento del tool prima dell'integrazione* (ci riferiamo in questo caso alle funzionalità dei test smell GF, ET, LCTM; con le relative statistiche di utilizzo del tool) e *introdurre i tre nuovi test smell*, ovvero MG, HCTD, TCD.

5.1 Impact Analysis

Per le stesse motivazioni espresse nel capitolo precedente nella sezione Impact Analysis, sono state riportate unicamente le modifiche dell'insieme *Starting Impact Set*, in quanto avendo a disposizione le documentazioni, la comprensione delle modifiche risulta essere semplice. Per questo motivo risulta essere che $|SIS| = |CIS| = |AIS|$.

5.1.1 Starting Impact Set - SIS

Artefatti	Impatto	Descrizione
Classi all'interno del package <i>main.java.testSmellDetection.detector</i> : 1. IDetector 2. TestSmellStructuralDetection 3. TestSmellTextualDetection	Debole	Aggiunta dei metodi per i nuovi test, sia nell'interfaccia (1) che nelle classi (2) e (3).
<i>main.java.commitWindowConstruction.commitWindowFactor</i>	Medio	Per la visualizzazione tramite UI dei nuovi test smell.

Classi all'interno del package <i>main.java</i> <i>.commitWindowConstruction.general</i> : 1. WarningWindow.java 2. RefactorWindow.java	Medio	Modifica della metodo WarningWindow nella classe (1) in modo da contenere tutti i test smell. Aggiunta di un costruttore nella classe (2) per la creazione dei pannelli per i nuovi test smell
Classi all'interno del package <i>main.java</i> <i>.action</i> : 1. StructuralDetectionAction 2. TextualDetectionAction	Medio	Aggiungere l'identificazione dei nuovi test smell nelle classi (1) e (2).
<i>java.extension.CommitFactory.java</i>	Medio	Aggiungere l'analisi automatica dei nuovi test smell.

A questo punto vengono riportate le cardinalità degli insiemi impattati. Per quanto detto precedentemente, gli insiemi avranno tutti la stessa cardinalità. Abbiamo quindi che:

- $|SIS| = |CIS| = |AIS| = 9$;
- $|FPIS| = |CIS| - |AIS| = 0$;

Avendo la cardinalità di tutti gli insiemi, andiamo a calcolare le metriche tradizionali valutare la bontà dell'Impact Analysis: *Recall* e *Precision*. Al fine di calcolare il valore di entrambe, dobbiamo dapprima calcolare la cardinalità dell'insieme intersezione fra *CIS* e *AIS*.

- $|CIS \cap AIS| = 9$;
- **Recall** = $\frac{|CIS \cap AIS|}{|AIS|} = \frac{9}{9} = 1$, questo vuol dire che non c'è stato nessun componente individuato durante la modifica che non sia stato anche individuato nel CIS.
- **Precision** = $\frac{|CIS \cap AIS|}{|CIS|} = \frac{9}{9} = 1$, valore conforme al fatto che *FPIS* è vuoto, in quanto la sua cardinalità è pari a 0.

Con questa nuova integrazione è stata incorporata la logica applicativa al fine di integrare al tool anche la ricerca e la rimozione dei tre test smell introdotti dalla repository di *fasanosalvatore/DARTS*. Si è deciso di mantenere la versione di Gradle della repository di *fasanosalvatore/DARTS* essendo quest'ultima più recente; con l'aggiunta del suffisso *snapshot* che garantisce sempre di puntare all'ultimo artefatto pubblicato.

5.2 Testing

Il testing di questa prima change request è stilato al **Capitolo 2 del Documento di Testing**.

5.3 Architettura del sistema

Dopo aver effettuato la seconda integrazione, l'architettura del sistema cambia, in quanto vengono inseriti all'interno del Tool anche nuove funzionalità circa la ricerca di ulteriori test smell. Vedendo le due immagini in Figura 5.1, possiamo notare come la differenza sostanziale fra le due architetture è proprio l'aggiunta dei tre nuovi test smell nel package *refactor*.

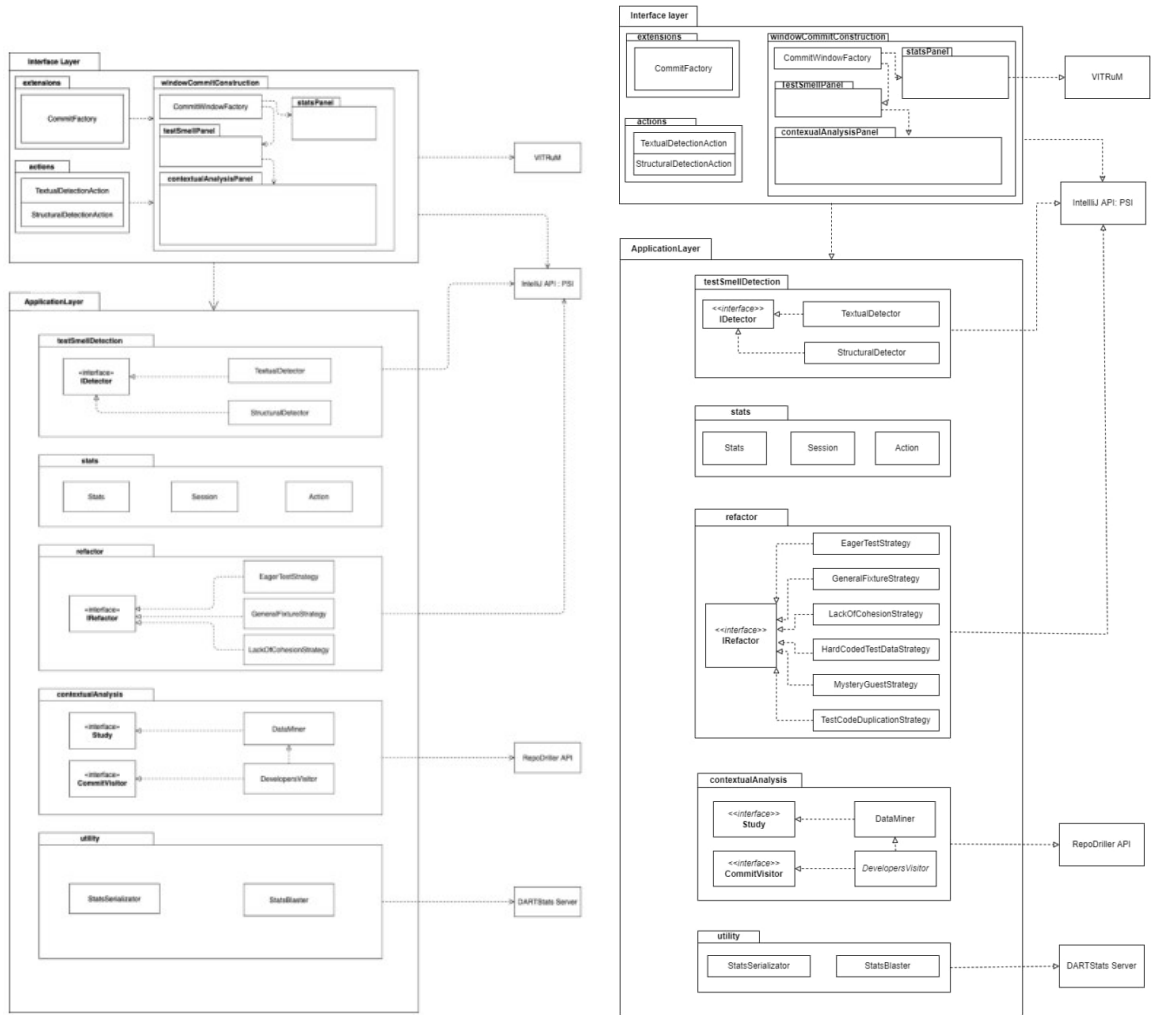
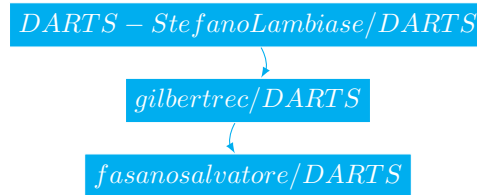


Figura 5.1: A sinistra l'architettura di DARTS dopo prima fase di integrazione, a destra la nuova architettura del sistema DARTS dopo la seconda fase di integrazione.

Capitolo 6

Terza Change Request $[CR_3]$



Una volta integrate correttamente le funzionalità della repository di *fasanosalvatore/DARTS*, il lavoro non è terminato in quanto il tool necessita di una fase di allineamento fra le due repository integrate. Ricordiamo che:

- la repository *gilbertrec/DARTS* ha introdotto nel tool il calcolo di alcune statistiche di utilizzo che siano esportabili;
- queste statistiche non sono presenti per quanto concerne i nuovi test smell introdotti dalla repository *fasanosalvatore/DARTS*;

Questa nuova fase di allineamento quindi ha lo scopo di estendere le funzionalità introdotte dalla repository *gilbertrec/DARTS*, alla repository *fasanosalvatore/DARTS*.

6.1 Impact Analysis

Prima di iniziare con l'allineamento di tutte le modifiche, dobbiamo capire quali sono gli artefatti che verranno impattati in questa fase. riportate unicamente le modifiche dell'insieme Starting Impact Set, in quanto avendo a disposizione le documentazioni e numerosi commenti all'interno dell'implementazione del server DARTStat, la comprensione delle modifiche risulta essere semplice. Per questo motivo risulta essere che $|SIS| = |CIS| = |AIS|$.

6.1.1 Starting Impact Set - SIS

Artefatto	Impatto	Descrizione
<i>windowCommitConstruction.statsPanel.StatsPanel.java</i>	Debole	UI per le visionare le statistiche dei test smell di <i>fasanosalvatore/DARTS</i> .

Alcune classi all'interno del package <i>main.java.stats</i> : 1. Action.java 2. Session.java	Medio	Allineamento della variabile enum nella classe (1) per tutti i test smell e l'implementazione nella classe (2) dei metodi per calcolare la densità, get e set per MG, HCTD e TCD.
Alcune classi all'interno del package <i>main.java.refactor.strategy</i> 1. HardCodedTestDataStrategy.java 2. MysteryGuestStrategy.java 3. TestCodeDuplicationStrategy.java	Debole	Aggiunta dei metodi <i>getActionForStats()</i> e <i>doAfterRefactor()</i> per ottenere dalla classe Action i risultati del test smell e infine l'azione viene aggiunta alla sessione.
Alcune classi all'interno del package <i>main.java.action</i> : 1. StructuralDetectionAction.java; 2. TextualDetectionAction.java;	Debole	Integrare i nuovi test smell introdotti dalla repository <i>fasanosalvatore/DARTS</i> nel calcolo delle statistiche in modalità strutturale (1) e in modalità Testuale (2).
<i>testSmellDetection.testSmellInfo.mysteryGuest.MysteryGuestInfo.java</i>	Medio	Aggiunti i metodi get e set.
<i>testSmellDetection.testSmellInfo.testCodeDuplication.TestCodeDuplicationInfo.java</i>	Medio	Aggiunti i metodi get e set.
Modifiche al server DARTStats lato <i>front-end</i> : 1. _id.vue; 2. SessionItem.vue;	Debole	Per permettere di visionare graficamente le statistiche dei tre nuovi smell.
Modifiche al server DARTStats lato <i>back-end</i> : 1. Session.js; 2. Stat.ccontroller.js;	Debole	Modifica della struttura dei modelli di database per permettere l'inserimento dei tre nuovi smell

A questo punto vengono riportate le cardinalità degli insiemi impattati. Per quanto detto precedentemente, gli insiemi avranno tutti la stessa cardinalità. Abbiamo quindi che:

- $|SIS| = |CIS| = |AIS| = 14$;
- $|FPIS| = |CIS| - |AIS| = 0$;

Avendo la cardinalità di tutti gli insiemi, andiamo a calcolare le metriche tradizionali valutare la bontà dell'Impact Analysis: *Recall* e *Precision*. Al fine di calcolare il valore di entrambe, dobbiamo dapprima calcolare la cardinalità dell'insieme intersezione fra *CIS* e *AIS*.

- $|CIS \cap AIS| = 14$;
- **Recall** = $\frac{|CIS \cap AIS|}{|AIS|} = \frac{14}{14} = 1$, questo vuol dire che non c'è stato nessun componente individuato durante la modifica che non sia stato anche individuato nel CIS.
- **Precision** = $\frac{|CIS \cap AIS|}{|CIS|} = \frac{14}{14} = 1$, valore conforme al fatto che *FPIS* è vuoto, in quanto la sua cardinalità è pari a 0.

6.2 Testing

Il testing di questa prima change request è stilato al **Capitolo 3** del *Documento di Testing*.

6.3 Architettura del sistema

La fase di integrazione non ha causato modifiche all'architettura del sistema vista in Figura 5.1 (immagine a destra).

Capitolo 7

Abbreviazioni

In questo capitolo vengono evidenziate le descrizioni di tutte le abbreviazioni utilizzate nel corso del documento.

CR Change Request;

RF Requisiti Funzionali;

RTM Matrice di Tracciabilità;

IA Impact Analysis;

SIS Start Impact Set;

CIS Candidate Impact Set;

DIS Discovered Impact Set;

AIS Actual Impact Set;

FPIS False Positive Impact Set;

GF General Fixture;

ET Eager Test;

LCTM Lack of Cohesion Test Method;

MG Mystery Guest;

HCTD Hard Coded Test Data;

TCD Test Code Duplication;

Bibliografia

- [1] Link alla repository di GitHub del progetto principale DARTS: <https://github.com/StefanoLambiase/DARTS>
- [2] Steafano Lambiase, Andrea Cupito, Fabiano Pecorelli, Andrea De Lucia and Fabio Palomba. *Just-In-Time Test Smell Detection and Refactoring: The DARTS Project*. In Proceedings of Seoul '20: ICPC International Conference on Program Comprehension (Seoul '20)
- [3] Link alla fork del progetto di **gilbertrec**: <https://github.com/gilbertrec/DARTS>
- [4] Link alla fork del progetto di **fasanosalvatore**: <https://github.com/fasanosalvatore/DARTS>