Università degli Studi di Salerno

Dipartimento di Informatica

Ingegneria, Gestione ed Evoluzione del Software

Statistiche sull'utilizzo del tool cASpER

Studenti:

Luca Maddaluno 0522500905

Gabriele Vassallo 0522501007

Prof. : Andrea De Lucia

Dott. : Emanuele Iannone

Dott. : Fabiano Pecorelli

Dott. : Manuel De Stefano

Revision History

Versione	Data	Autore	Descrizione
1.0	15/05/2021	Gabriele Vassallo	Creazione del documento
1.1	20/05/2021	Gabriele Vassallo, Luca Maddaluno	Aggiunte change request
1.2	24/05/2021	Gabriele Vassallo	Aggiunto reverse engineering
1.3	03/06/2021	Gabriele Vassallo, Luca Maddaluno	Aggiunto impact analysis
1.4	10/06/2021	Gabriele Vassallo, Luca Maddaluno	Aggiunto testing, revisionato impact analysis
1.4.1	16/06/2021	Luca Maddaluno	Riviste alcune imprecisioni nel capitolo di test
1.5	19/06/2021	Gabriele Vassallo, Luca Maddaluno	Revisione finale

1 Introduzione

Il plugin cASpER, per IntelliJ IDEA, è in grado di ispezionare tutti i file sorgente Java per identificare Code Smell e proporre per ciascuno (a seconda della tipologia) un refactoring per rimuoverlo. Dopo l'installazione, che può essere effettuata tramite l'apposita sezione di IntelliJ, il tool può essere lanciato manualmente dall'utente tramite il pulsante "Tools" della barra dei menù dell'IDE. L'interfaccia è solamente grafica ed implementata con la classica Java Swing.
Dal menù a tendina è possibile richiamare due differenti pannelli, uno dedicato all'utilizzo del tool e l'altro dedicato alla configurazione.
Il pannello di configurazione permette di settare i parametri per la rilevazione di ognuno dei quattro smell che attualmente cASpER riesce ad identificare e risolvere facendo un'analisi statica basata sugli AST (tramite la libreria Intellij PSI).
Il pannello di analisi mostra i risultati dell'analisi che viene effettuata dal tool prima che il pannello sia visualizzato. Il pannello mostra la lista degli smell identificati e permette di filtrare tale lista tramite alcune checkbox e parametri. Da questo pannello è possibile avviare l'ispezione di uno smell. Questa operazione porta l'utente su un successivo pannello, tramite il quale, è possibile richiedere la risoluzione automatica dello smell.

1.2 Obiettivo del progetto

Aggiungere un sottosistema al tool cASpER che permetta di calcolare e visualizzare alcune statistiche di utilizzo del tool. Ad ogni esecuzione del tool saranno salvati alcuni dati. I dati salvati saranno relativi a informazioni sull'utilizzo e il funzionamento di cASpER. I dati raccolti saranno visualizzabili in una schermate dedicate, dalla quale sarà anche possibile esportare su file i dati raccolti. Le informazioni statistiche così ottenute, in futuro, potranno essere sfruttate come riferimento per la pianificazione di interventi di manutenzione e aggiornamento.

1.3 Change Request

In tabella il set di change request individuato:

Change Request	
M1	Tempo di elaborazione
M2	Numero di smell identificati
M3	Analisi senza refactoring
M4	Tempo di utilizzo
M5	Numero di volte in cui il tool non trova soluzioni

(M1) Tempo di elaborazione
Memorizza il tempo impiegato dal tool per completare un analisi.

(M2) Numero totale di smell identificati
Calcola il numero di smell totali identificati dal tool, distinguendoli per tipologia.

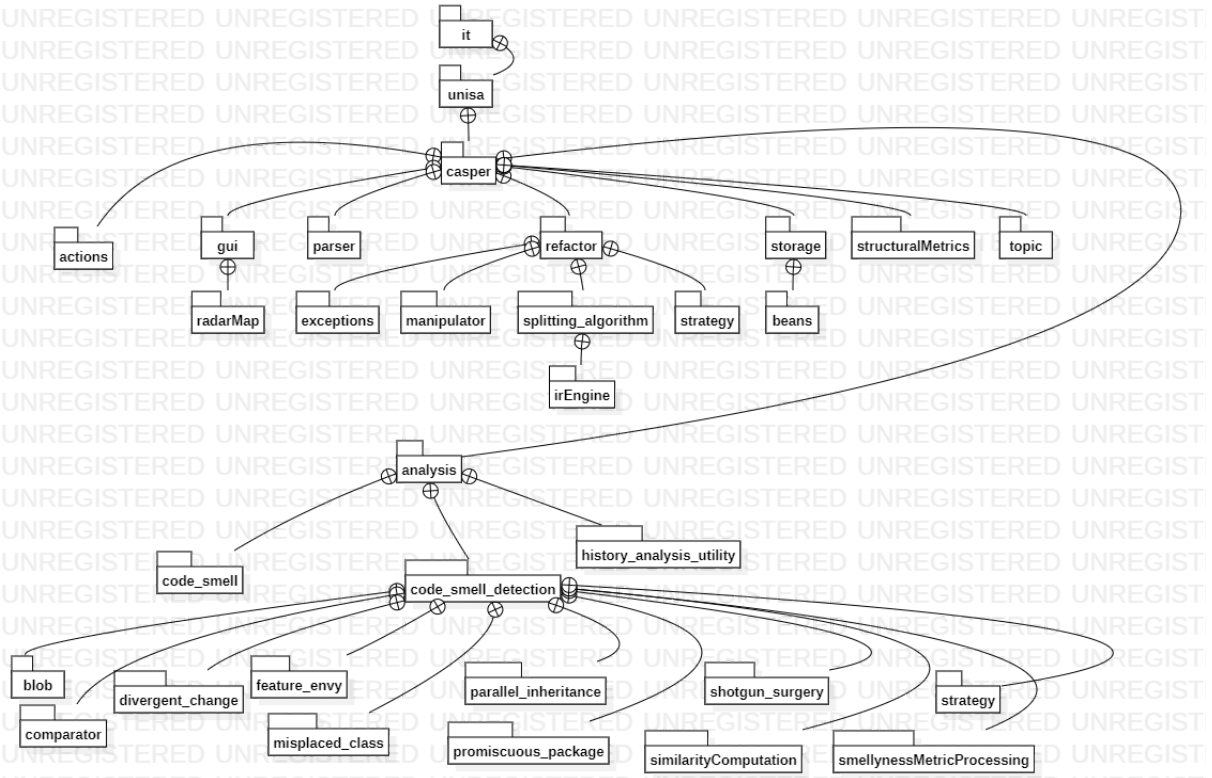
(M3) Analisi senza Refactoring
Memorizza il numero di volte in cui l'utente lancia un'analisi ma poi decide di non utilizzare il tool per cercare una soluzione di refactoring.

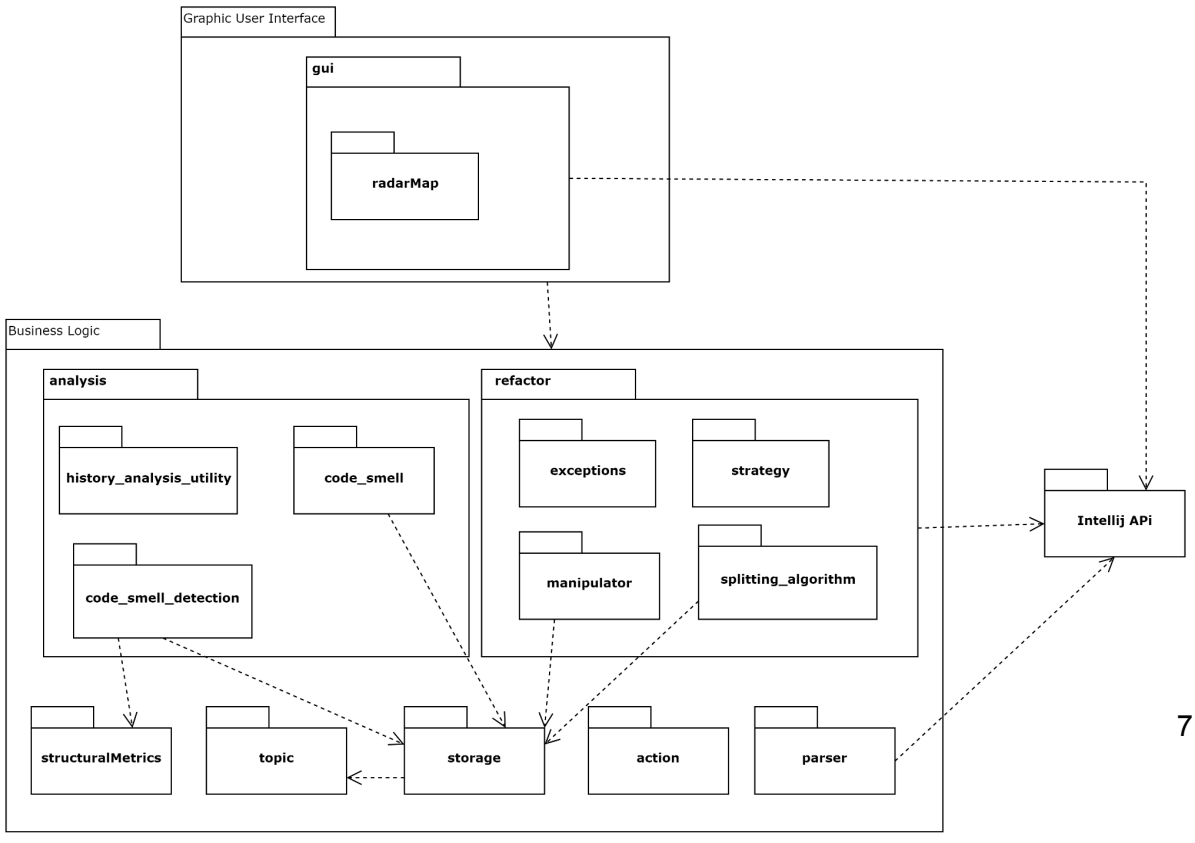
(M4) Tempo di utilizzo
Memorizza il tempo di utilizzo dell'interfaccia del tool.

(M5) Numero di volte in cui il tool non trova soluzioni
Memorizza il numero di volte in cui, al termine di un'analisi, il tool identifica degli smell ma non riesce a trovare una soluzione.

2 Reverse engineering

Per comprende e familiarizzare con codice di cASpER è stato necessario compiere alcune operazioni di reverse engineering. Durante questa fase, con l'ausilio del tool "StarUML" è stato possibile ricavare diagrammi UML con la gerarchia dei pacchetti.





Successivamente abbiamo ispezionato i singoli pacchetti e, dopo aver visionato le classi presenti in ciascun pacchetto, abbiamo descritto in modo sintetico il contenuto di ogni pacchetto. Le descrizioni sono riportate di seguito.

2.1 Packages map

```
src/
├── main/
│   └── java/
│       └── it.unisa.casper/
│           ├── actions/
│           ├── analysis/
│           │   ├── code_smell/
│           │   ├── code_smell_detection/
│           │   │   ├── blob/
│           │   │   ├── comparator/
│           │   │   ├── divergent_change/
│           │   │   ├── feature_envy/
│           │   │   ├── misplaced_class/
│           │   │   ├── parallel_inheritance/
│           │   │   ├── promiscuous_package/
│           │   │   ├── shotgun_surgery/
│           │   │   ├── similarityComputation/
│           │   │   ├── smellynessMetricProcessing/
│           │   │   └── strategy/
│           │   └── history_analysis_utility/
│           ├── gui/
│           │   └── radarMap/
│           ├── parser/
│           ├── refactor/
│           │   ├── exceptions/
│           │   ├── manipulator/
│           │   ├── splitting_algorithm/
│           │   │   └── irEngine/
│           │   └── strategy/
│           ├── statistics/
│           ├── storage.beans/
│           ├── structuralMetrics/
│           └── topic/
└── test/
    └── it.unisa.casper/
        ├── analysis/
        │   └── code_smell_detection/
        │       ├── blob/
        │       ├── feature_envy/
        │       ├── Helper/
        │       ├── misplaced_class/
        │       └── promiscuos_package/
        └── refactor/
```

2.1.1 Pacchetti presenti nella cartella main

Contengono codice di produzione. Tutti i pacchetti elencati sono presenti nel pacchetto principale `it.unisa.casper`.

Package `actions`
Le classi nel pacchetto `actions` si occupano del funzionamento principale del tool: avvio, configurazione delle threshold, esecuzione e visualizzazione della GUI.

Package `analysis`
Si occupa di tutta l'analisi del codice per trovare gli smell. I suoi sottopacchetti sono specializzati nei diversi tipi di smell e ognuno si occupa di analizzare, istanziare e gestire gli smell.

Package `code_smell`
Comprende tutte le classi che rappresentano l'oggetto Code Smell, necessarie per tenere traccia degli smell individuati durante l'analisi del codice.

Package `code_smell_detection`
I sottopacchetti riguardanti gli smell si occupano di gestire l'analisi testuale e strutturale degli stessi, oltre che l'analisi storica. I pacchetti rimanenti gestiscono il calcolo delle metriche necessarie all'individuazione dei code smell.

Subpackage `blob`
Contiene le classi la gestione dei code smell di tipo Blob.

Subpackage `comparator`
Comprende le classi che forniscono metodi di comparazione tra bean diversi.

Subpackage `divergent_change`
Contiene le classi la gestione dei code smell di tipo Divergent Change.

Subpackage `feature_envy`
Contiene le classi la gestione dei code smell di tipo Feature Envy.

Subpackage `misplaced_class`
Contiene le classi la gestione dei code smell di tipo Misplaced Class.

Subpackage `parallel_inheritance`
Contiene le classi la gestione dei code smell di tipo Parallel Inheritance.

Subpackage `promiscuous_package`
Contiene le classi la gestione dei code smell di tipo Promiscuous Package.

Subpackage `shotgun_surgery`
Contiene le classi la gestione dei code smell di tipo Shotgun Surgery.

Subpackage `similarityComputation`

Calcola la similarità tra le classi basata sui termini presenti al loro interno.

Subpackage `smellynessMetricProcessing`
Si occupa di processare le metriche di similarità individuate.

Subpackage `strategy`
Comprende le interfacce da implementare per la detection dei code smell.

Package `history_analysis_utility`
Nel pacchetto sono presenti le classi che contengono script Python per l'individuazione degli smell e delle loro priorità.

Package `gui`
Classi che si occupano di creare e visualizzare la GUI del tool.

Subpackage `radarMap`
Contiene le classi per la creazione e visualizzazione delle `radarMap`.

Package `parser`
Si occupa della creazione e del parsing di diversi tipi di bean.

Package `refactor`
Comprende diversi pacchetti utili al refactoring del codice.

Package `exceptions`
Introduce diversi tipi di eccezioni specifiche.

Subpackage `manipulator`
Classi che si occupano di effettuare la parte di refactoring che comprende il cambiamento del codice, la sua sostituzione e lo spostamento di parti di codice.

Subpackage `splitting_algorithm`
Si occupa della parte di refactoring che riguarda lo splitting di classi e pacchetti.

Subpackage `irEngine`
Riguarda strategie di gestione interne.

Subpackage `strategy`
Sono istanziate le classi responsabili del refactoring.

Package `statistics`
Si occupa della raccolta delle statistiche relative al tool, del salvataggio e dell'esportazione su file esterni.

Package `storage.beans`
Contiene le classi che rappresentano bean per classi, metodi, pacchetti e variabili.

Package `structuralMetrics`
Classi che gestiscono le metriche di Chidamber-Kemerer analizzate da visualizzare, come righe di codice o numero di classi.

Package `topic`
Estrae i topic, ovvero i termini più frequenti, dal contenuto testuale di un bean per classi, metodi e pacchetti.

2.1.2 Pacchetti presenti nella cartella test

Contengono codice di test. Tutti i pacchetti elencati sono presenti nel pacchetto principale `it.unisa.casper`.

Package `analysis`
Comprende classi di test per le classi che si occupano di analisi del codice.

Package `code_smell_detection`
Contiene test per l'individuazione dei diversi tipi di code smell.

Subpackage `blob`
Classi che testano il funzionamento testuale e strutturale dei code smell di tipo Blob.

Subpackage `feature_envy`
Classi che testano il funzionamento testuale e strutturale dei code smell di tipo Feature Envy.

Subpackage `Helper`
Comprende classi che generano valori specifici utilizzati nei test.

Subpackage `misplaced_class`
Classi che testano il funzionamento testuale e strutturale dei code smell di tipo Misplaced Class.

Subpackage `promiscuous_package`
Classi che testano il funzionamento testuale e strutturale dei code smell di tipo Promiscuous Package.

Package `refactor`
Comprende classi di test per le classi che si occupano del refactoring del codice.

3 Impact Analysis

Le operazioni di reverse engineering ci hanno permesso di avere un quadro chiaro sulla struttura del codice e una conoscenza sufficientemente approfondita sul tipo di classi contenute in ogni pacchetto.
Sulla base della conoscenza acquisita durante la fase di reverse engineering è stato possibile realizzare un diagramma di attività per ciascuna change request. I diagrammi di attività, realizzati ad alto livello, ci hanno permesso di individuare le classi coinvolte in ciascuna modifica e stimare in numero di classi che sarebbe stato necessario aggiungere al sistema. Questa operazione ci ha permesso di individuare uno starting impact set che abbiamo poi rappresentato con l'ausilio di grafi che evidenziassero le relazioni presenti tra i work product.
Abbiamo poi individuato il candidate impact set basandoci sulle dipendenze sintattiche tra le classi. Per svolgere questa attività abbiamo utilizzato delle matrici delle dipendenze. Per ottenere un candidate impact set con un numero quanto più basso possibile di falsi positivi e falsi negativi abbiamo utilizzando l'approccio basato sulla distanza e ci siamo avvalsi della nostra esperienza e conoscenza del codice.
Dopo aver effettuato le modifiche necessarie per l'implementazione delle change request abbiamo realizzato un grafo per rappresentare l'actual impact set.

3.1 Approccio

In questa sezione viene definito l'approccio seguito per l'Impact Analysis. Per ogni change request vengono effettuate delle operazioni comuni per individuare le classi potenzialmente coinvolte nelle modifiche:

- la funzionalità viene rappresentata ad alto livello tramite un diagramma di attività;
- viene individuato lo Starting Impact Set con la costruzione di un grafico di tracciabilità;
- viene individuato il Candidate Impact Set tramite costruzione di matrice di dipendenza, con approccio basato sulla distanza e sulla conoscenza del codice sorgente;
- viene quindi trovato l'Actual Impact Set. Nel grafico di tracciabilità sono indicati in grigio i work product aggiunti.
- vengono calcolate le metriche di *Precision*, *Recall* e *Adequacy*.

3.2 (M1) Tempo di elaborazione

Memorizza il tempo impiegato dal tool per completare un'analisi.

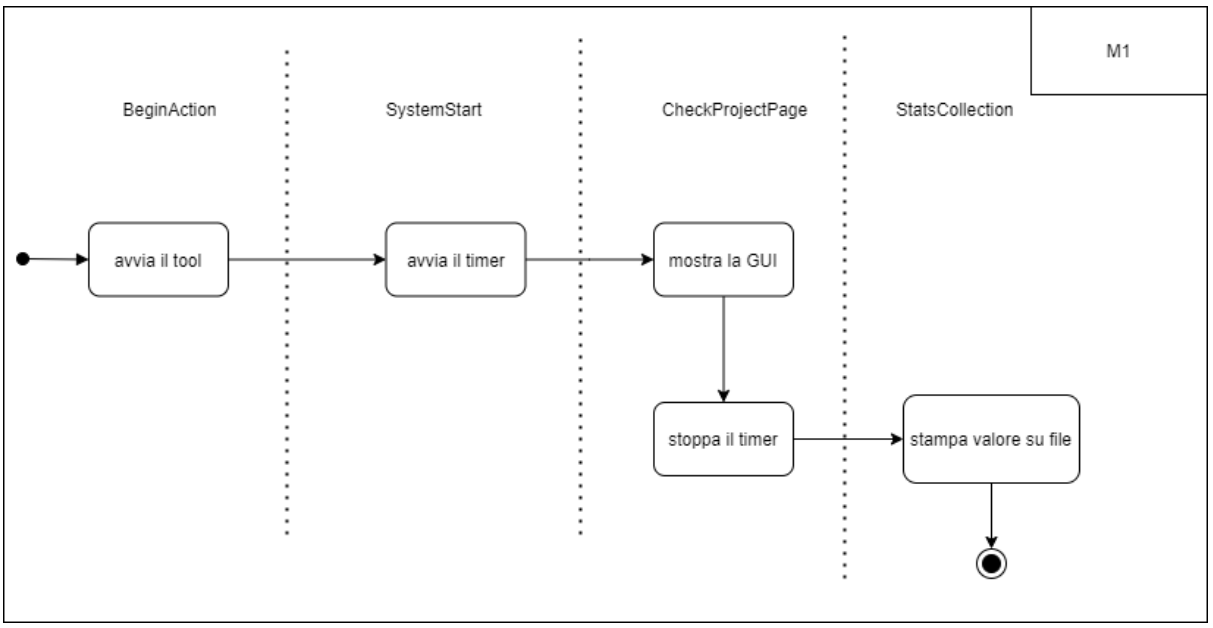
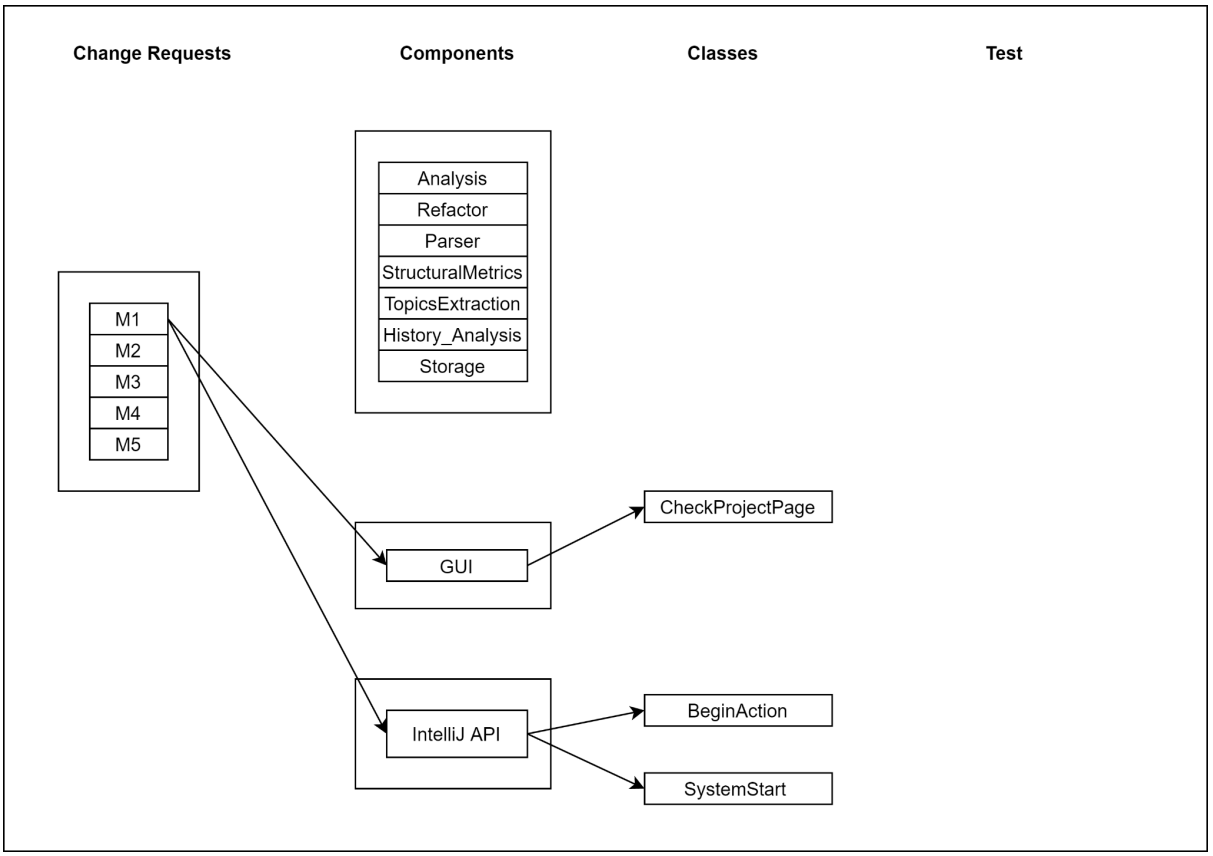


Diagramma di attività (M1)



Starting impact set (M1)

Nella costruzione della matrice delle dipendenze sono state prese in considerazione le classi con distanza 1. Tra queste si è notato che un gruppo di classi vengono istanziate sempre nello stesso modo e sono legate tra di loro quando vengono effettuate operazioni che modificano le classi stesse: ClassBean, MethodBean, PackageBean. Essendo l'obiettivo della change request completamente distante dalle operazioni di queste classi, si conclude che non saranno

incluse nel Candidate Impact Set. Inoltre la classe HistoryAnalysisStartup viene utilizzata solo come utility accessoria, per cui non verrà mai presa in considerazione.

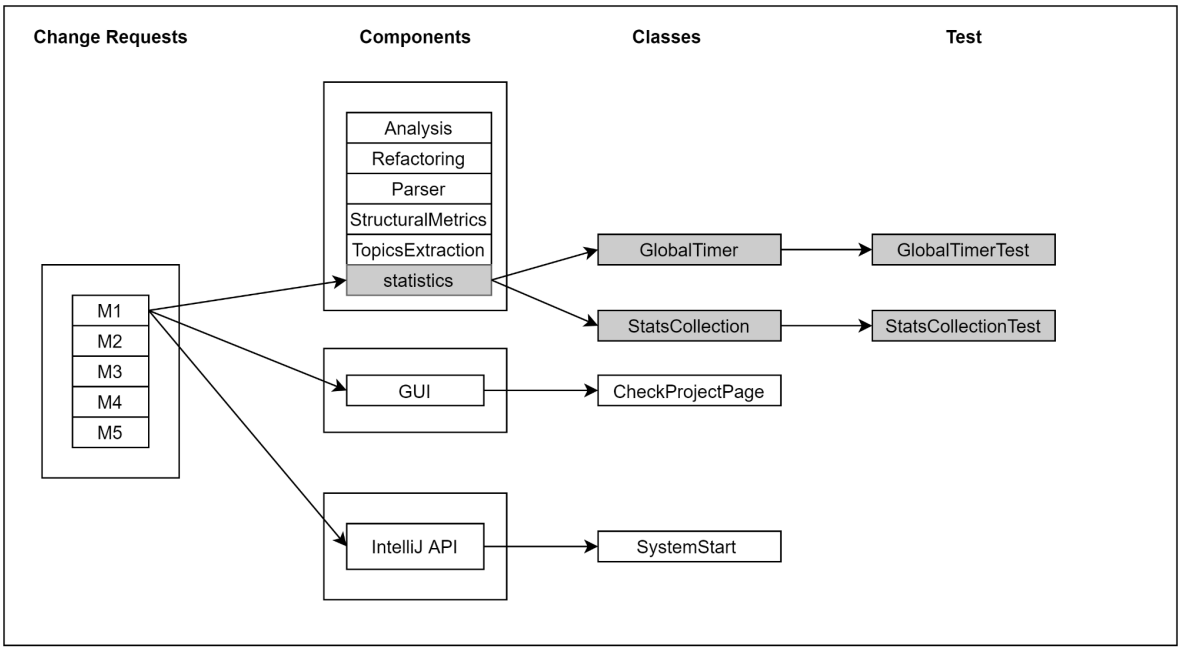
Matrice dipendenze M1	BeginAction	System Start	CheckProgectPage	CodeSmell	HistoryAnalysisStartup	ClassBean	MethodBean	PackageBean	CKMetrics	CheckProjectPage	PsiParser
BeginAction											
System Start											
CheckProjectPage											
CodeSmell			1		2	2	2	2	2		
HistoryAnalysisStartup		1	1								
ClassBean		1	1								
MethodBean		1	1						2		
PackageBean		1	1						2		
CKMetrics			1								
CheckProjectPage		1									
PsiParser		1									

Candidate impact set

Il Candidate Impact Set è dunque formato dalle seguenti classi:

- CodeSmell
- CKMetrics
- CheckProjectPage
- PsiParser
- SystemStart

Dopo aver effettuato le modifiche, l'Actual Impact Set è mostrato nel grafico seguente:



Actual impact set

Le metriche individuate per M1 sono indicate di seguito:

$$|FPIS| = |CIS \setminus AIS| = 5 - 2 = 3$$

$$Recall = \frac{|CIS \cap AIS|}{|AIS|} = 2/2 = 1$$

$$Precision = \frac{|CIS \cap AIS|}{|CIS|} = 2/5 = 0.4$$

$$Inclusiveness = 1$$

La modifica è stata effettuata con successo. Durante il processo di Impact Analysis c'è stata una sovrastima del Candidate Impact Set come si può vedere dalla *precision*, tuttavia la *recall* è pari a 1, per cui non ci sono classi che la stima non ha considerato.

3.3 (M2) Numero totale di smell identificati

Calcola il numero di smell totali identificati dal tool, distinguendoli per tipologia.

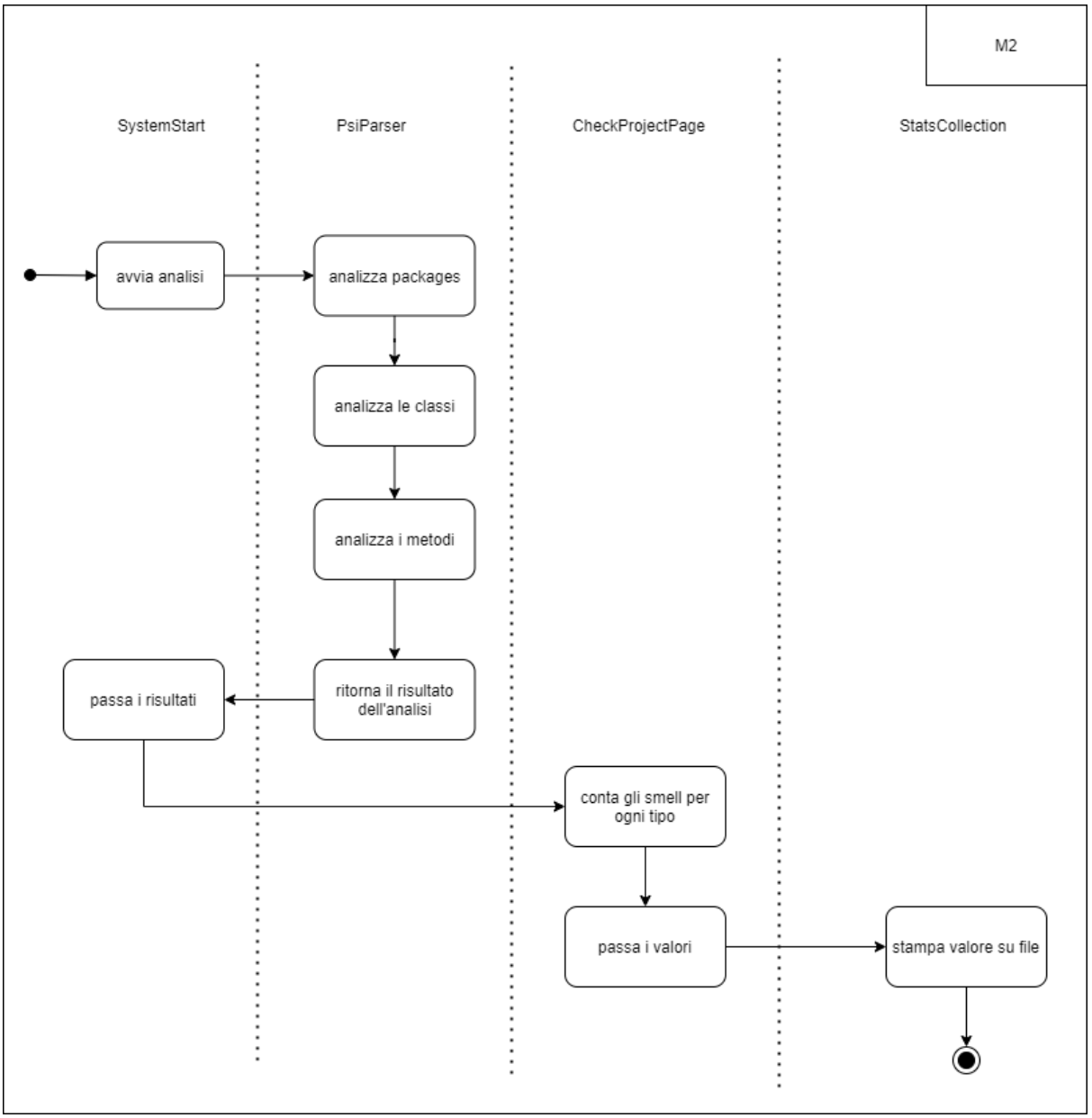
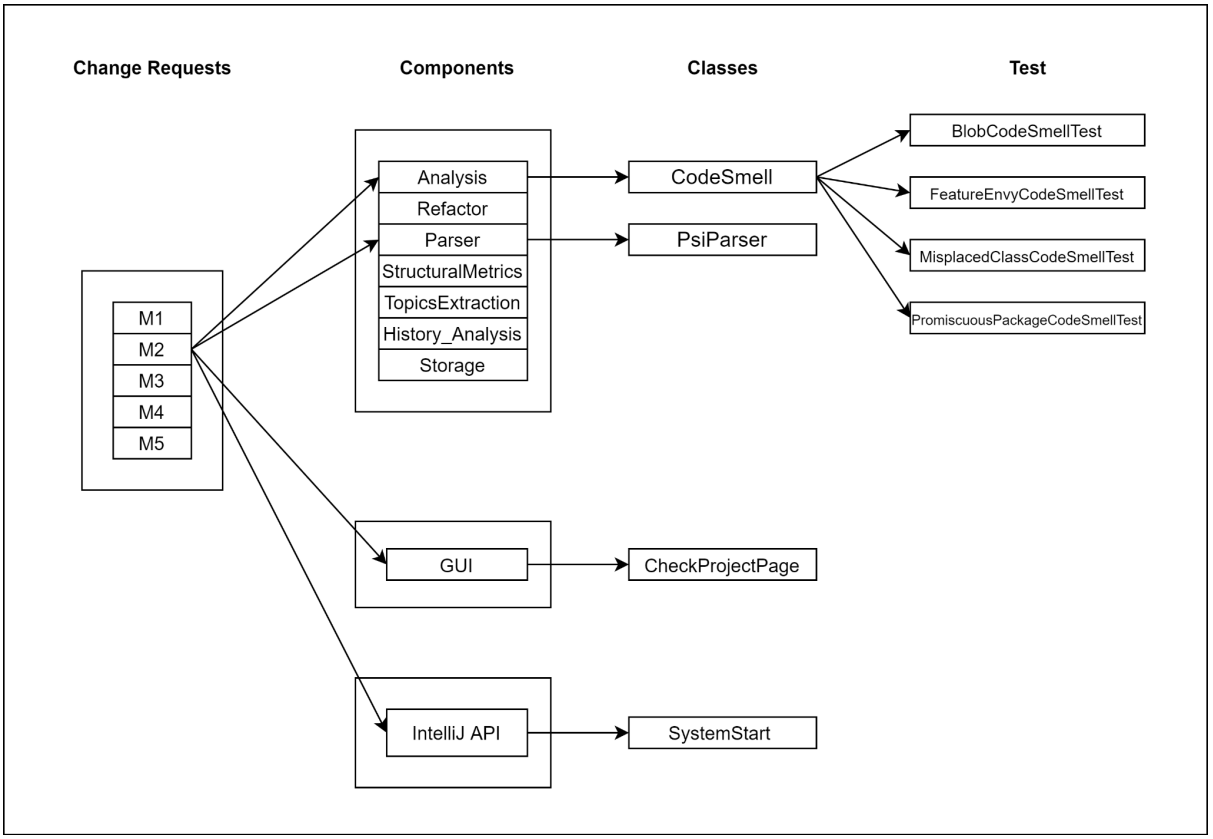


Diagramma di attività (M2)



Starting impact set (M2)

Nella costruzione della matrice delle dipendenze sono state prese in considerazione le classi con distanza 1. Tra queste si è notato che un gruppo di classi vengono istanziate sempre nello stesso modo e sono legate tra di loro quando vengono effettuate operazioni che modificano le classi stesse: ClassBean, MethodBean, PackageBean. Essendo l'obiettivo della change request completamente distante dalle operazioni di queste classi, si conclude che non saranno incluse nel Candidate Impact Set.

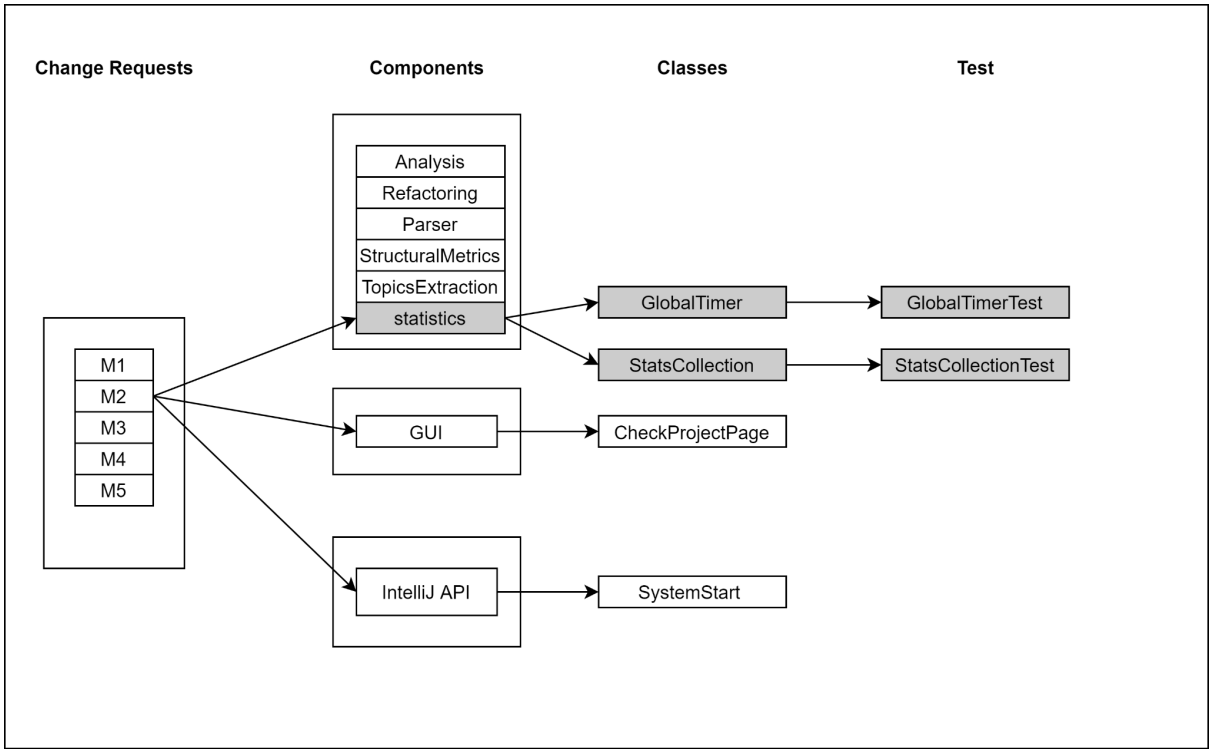
Matrice dipendenze M2	System Start	CheckProjectPage	CodeSmell	PsiParser	HistoryAnalysisStartup	ClassBean	MethodBean	PackageBean	CKMetrics
System Start									
CheckProjectPage									
CodeSmell		1			2	2	2		2
PsiParser	1								
HistoryAnalysisStartup	1	1							
ClassBean	1	1			2				
MethodBean	1	1							2
PackageBean	1	1							2
CKMetrics		1							

Candidate impact set (M2)

Il Candidate Impact Set è dunque formato dalle seguenti classi:

- CodeSmell
- PsiParser
- CheckProjectPage
- SystemStart

Dopo aver effettuato le modifiche, l'Actual Impact Set è mostrato nel grafico seguente:



Actual impact set (M2)

Le metriche individuate per M2 sono indicate di seguito:

$$|FPIS| = |CIS \setminus AIS| = 4 - 2 = 2$$

$$Recall = \frac{|CIS \cap AIS|}{|AIS|} = 2/2 = 1$$

$$Precision = \frac{|CIS \cap AIS|}{|CIS|} = 2/4 = 0.5$$

$$Inclusiveness = 1$$

La modifica è stata effettuata con successo. Anche qui si è verificata una sovrastima del Candidate Impact Set come si può vedere dalla *precision*, tuttavia la *recall* è pari a 1, quindi non ci sono classi modificate che sono state tralasciate dalla stima.

3.4 (M3) Analisi senza Refactoring

Memorizza il numero di volte in cui l'utente lancia un'analisi ma poi decide di non utilizzare il tool per cercare una soluzione di refactoring.

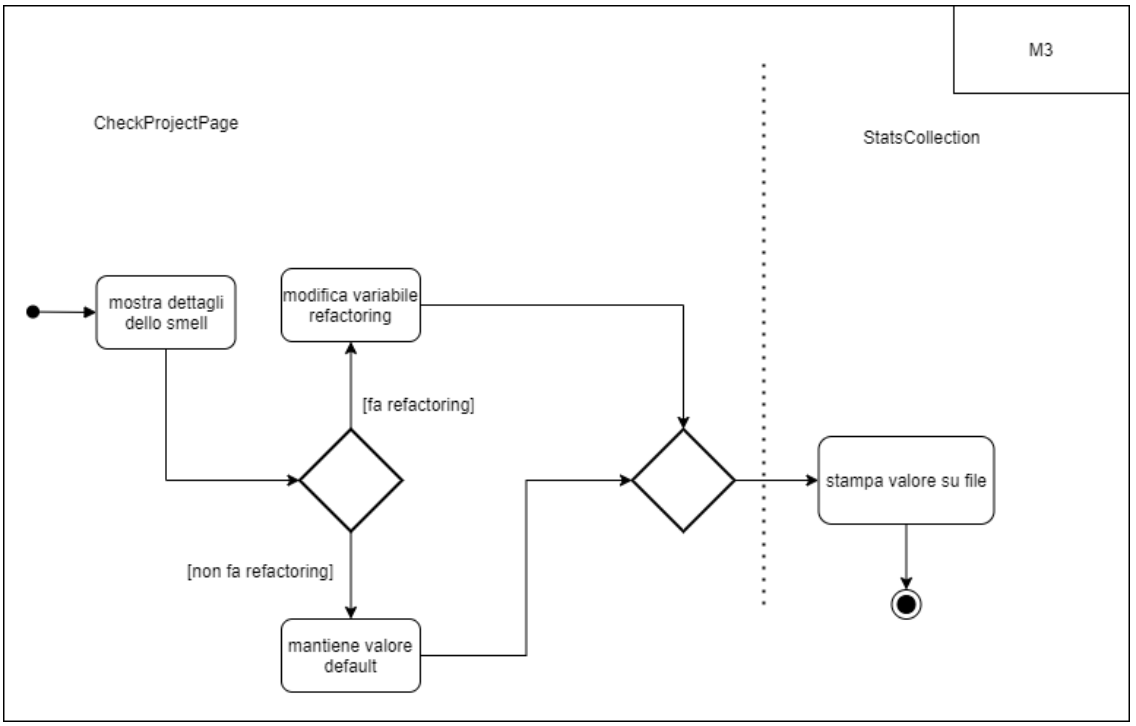
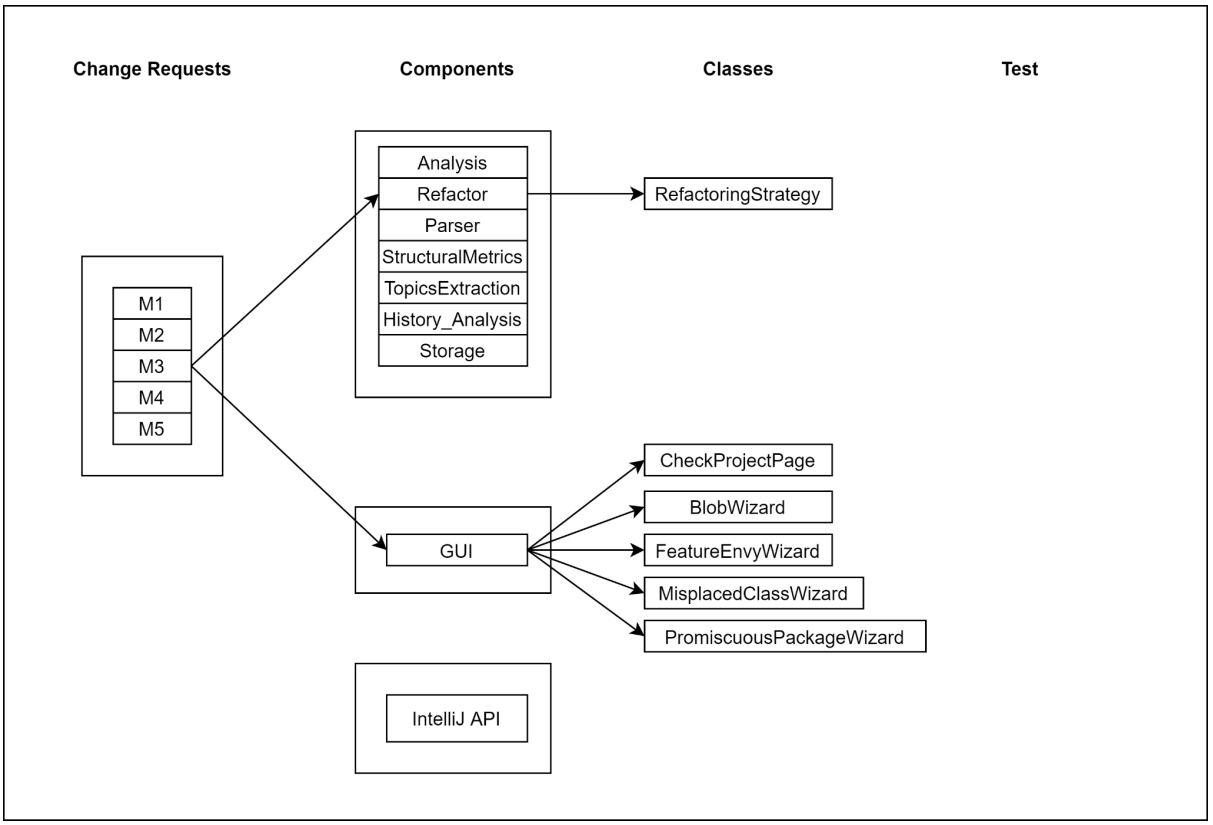


Diagramma di attività (M3)



Starting impact set (M3)

Nella costruzione della matrice delle dipendenze sono state prese in considerazione le classi con distanza 1. Anche qui è presente il gruppo di classi legate tra di loro: ClassBean, MethodBean, PackageBean. Essendo l'obiettivo della change request completamente distante dalle operazioni di queste classi, si conclude che non saranno incluse nel Candidate Impact Set. Inoltre sono presenti le classi RefactoringStrategy e RefactoringManager che vengono chiamate in causa nel processo di refactoring, per cui non saranno prese in considerazione.

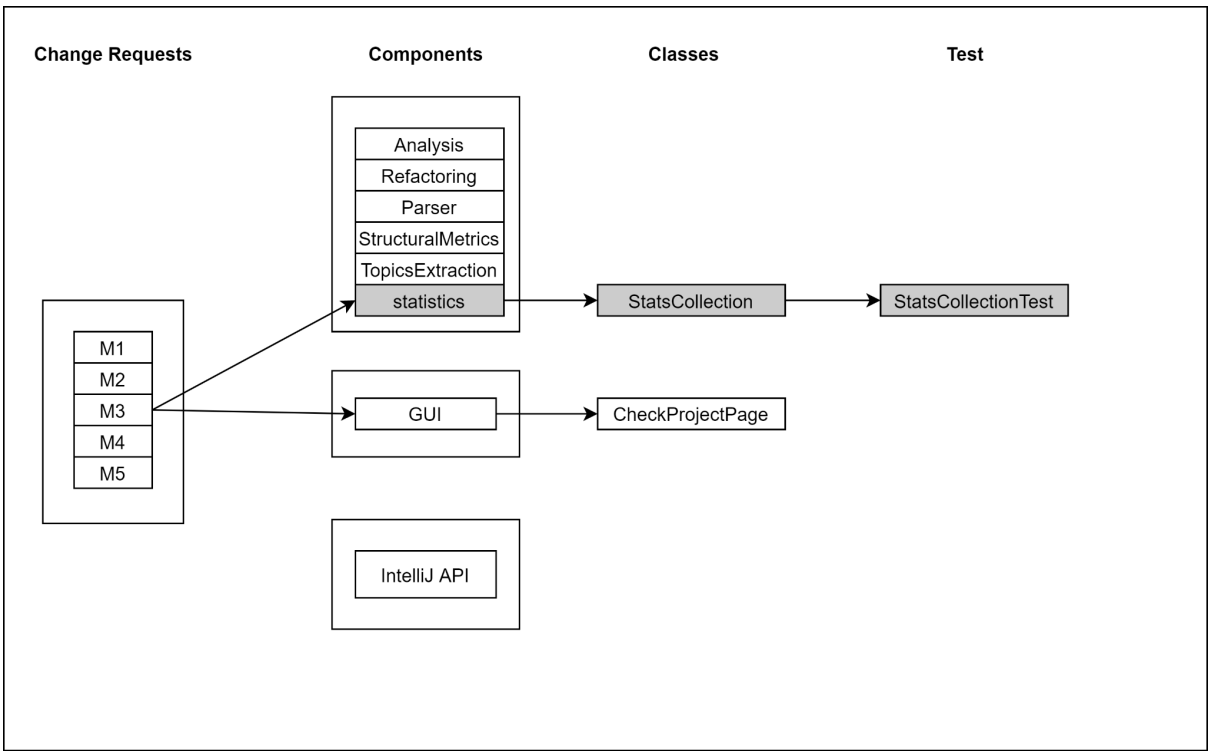
Matrice dipendenze M3	CheckProjectPage	BlobWizard	FeatureEnvyWizard	MisplacedClassWizard	PromiscuousPackageWizard	RefactoringStrategy	CodeSmell	HistoryAnalysisStartup	ClassBean	MethodBean	PackageBean	CKMetrics	RefactoringStrategy	RefactoringManager	InstanceVariableBean	UpdateClassUtility	TopicExtracter
CheckProjectPage																	
BlobWizard																	
FeatureEnvyWizard																	
MisplacedClassWizard																	
PromiscuousPackageWizard																	
RefactoringStrategy																	
CodeSmell	1							2	2	2	2						
HistoryAnalysisStartup	1																
ClassBean	1		1	1	1							2				2	2
MethodBean	1	1	1		1							2					2
PackageBean	1		1	1	1							2					2
CKMetrics	1	1			1												
RefactoringStrategy		1	1		1												
RefactoringManager		1	1	1	1												
InstanceVariableBean		1			1							2					
UpdateClassUtility			1														
TopicExtracter			1	1													

Candidate impact set (M3)

Il Candidate Impact Set è dunque formato dalle seguenti classi:

- CheckProjectPage
- CodeSmell
- InstanceVariableBean
- UpdateClassUtility
- TopicExtracter

Dopo aver effettuato le modifiche, l'Actual Impact Set è mostrato nel grafico seguente:



Actual impact set (M3)

Le metriche individuate per M3 sono indicate di seguito:

$$|FPIS| = |CIS \setminus AIS| = 5 - 1 = 4$$

$$Recall = \frac{|CIS \cap AIS|}{|AIS|} = 1/1 = 1$$

$$Precision = \frac{|CIS \cap AIS|}{|CIS|} = 1/5 = 0.2$$

$$Inclusiveness = 1$$

La *precision* molto bassa è dovuta ad una sovrastima del Candidate Impact Set; tuttavia la modifica è stata effettuata con successo e senza problemi grazie al valore di *recall* pari a 1.

3.5 (M4) Tempo di utilizzo

Memorizza il tempo di utilizzo dell'interfaccia del tool.

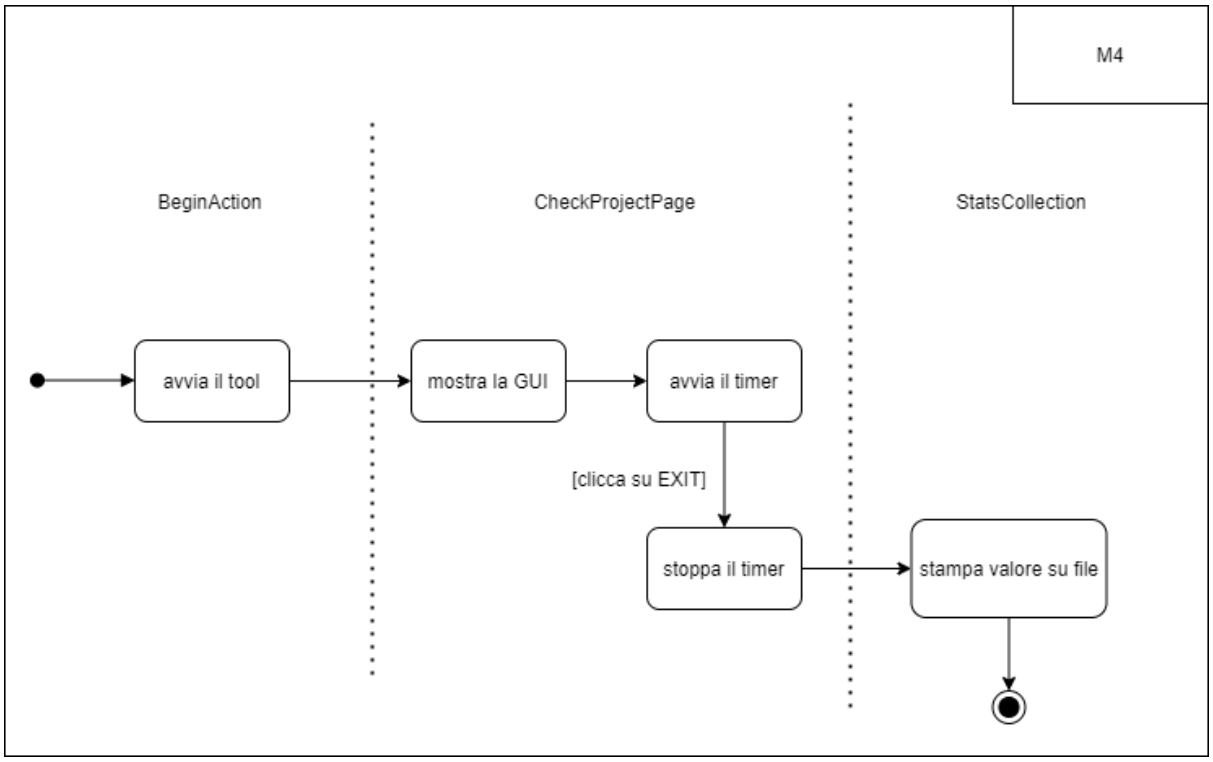
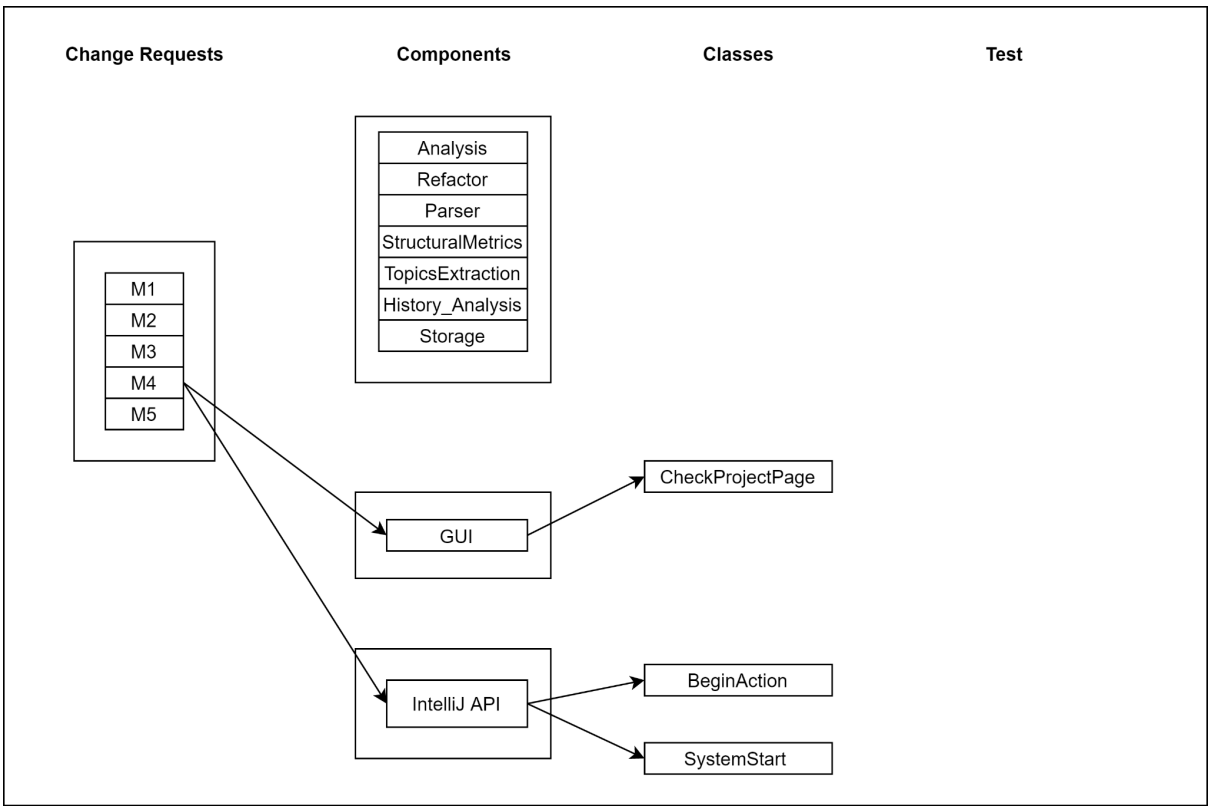


Diagramma di attività (M4)



Starting impact set (M4)

Nella costruzione della matrice delle dipendenze sono state prese in considerazione le classi con distanza 1. L'analisi di questa change request è molto simile alla CR M1, con la differenza che SystemStart alla fine non verrà incluso nel set finale in quanto la funzionalità non richiede di essere attiva appena il tool viene avviato.

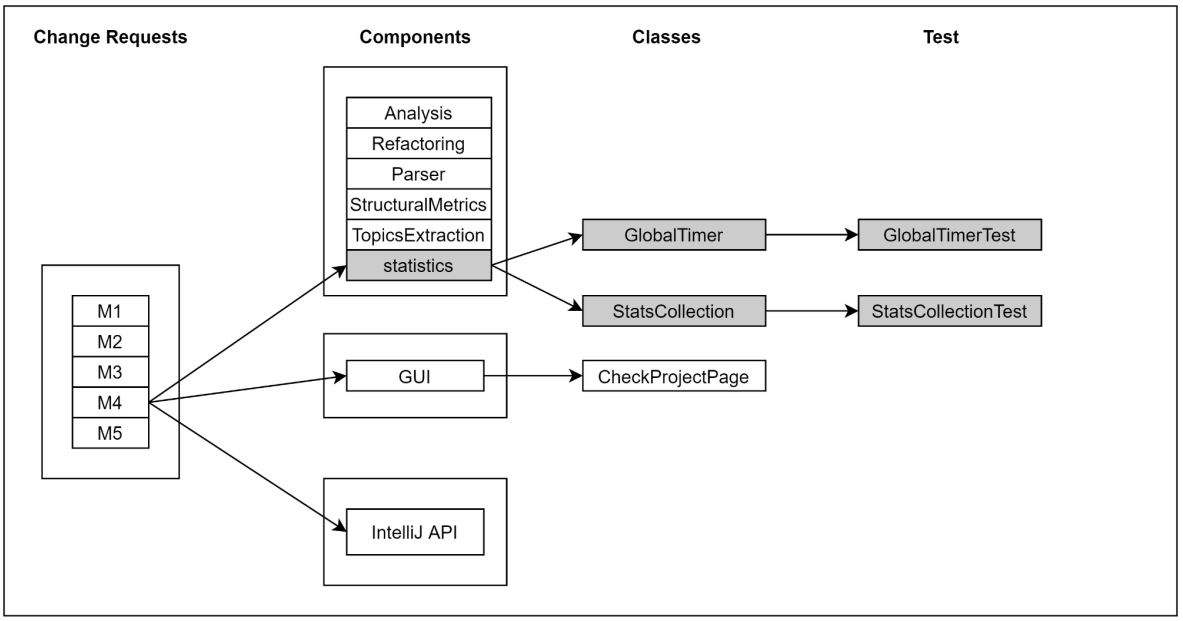
Matrice dipendenze M4	CheckProjectPage	BeginAction	SystemStart	CodeSmell	HistoryAnalysisStartup	ClassBean	MethodBean	PackageBean	CKMetrics	PsiParser
CheckProjectPage			1							
BeginAction										
SystemStart										
CodeSmell	1				2	2	2	2		
HistoryAnalysisStartup	1									
ClassBean	1								2	2
MethodBean	1								2	2
PackageBean	1		1						2	2
CKMetrics	1									
PsiParser			1							

Candidate impact set (M4)

Il Candidate Impact Set è dunque formato dalle seguenti classi:

- CodeSmell
- CKMetrics
- CheckProjectPage
- PsiParser
- SystemStart

Dopo aver effettuato le modifiche, l'Actual Impact Set è mostrato nel grafico seguente:



Actual impact set (M4)

$$|FPIS| = |CIS \setminus AIS| = 5 - 1 = 4$$

$$Recall = \frac{|CIS \cap AIS|}{|AIS|} = 1/1 = 1$$

$$Precision = \frac{|CIS \cap AIS|}{|CIS|} = 1/5 = 0.2$$

$$Inclusiveness = 1$$

La *precision* molto bassa è dovuta ad una sovrastima del Candidate Impact Set; tuttavia la modifica è stata effettuata con successo e senza problemi grazie al valore di *recall* pari a 1.

3.6 (M5) Numero di volte in cui il tool non trova soluzioni

Memorizza il numero di volte in cui, al termine di un'analisi, il tool identifica degli smell ma non riesce a trovare una soluzione.

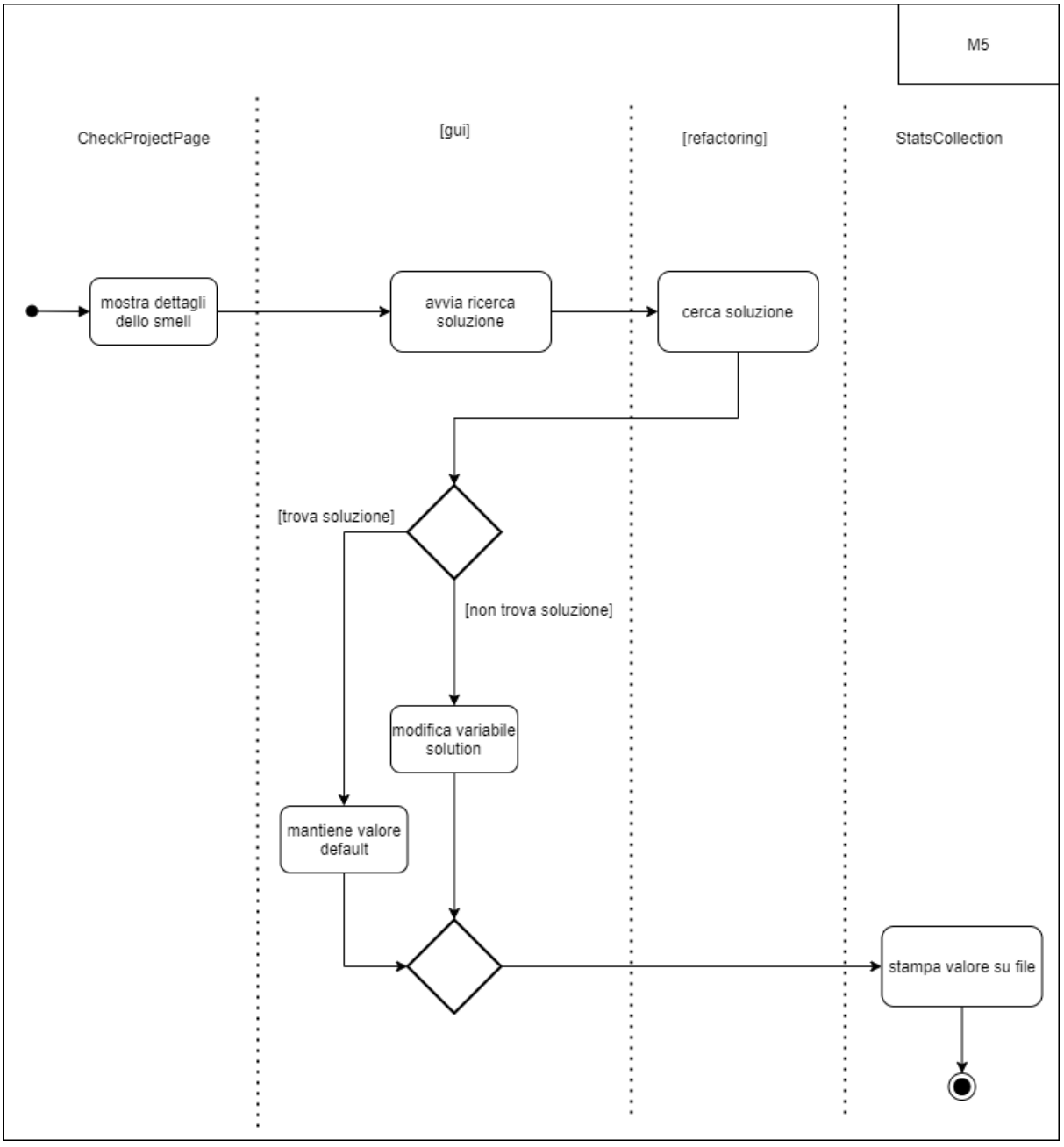
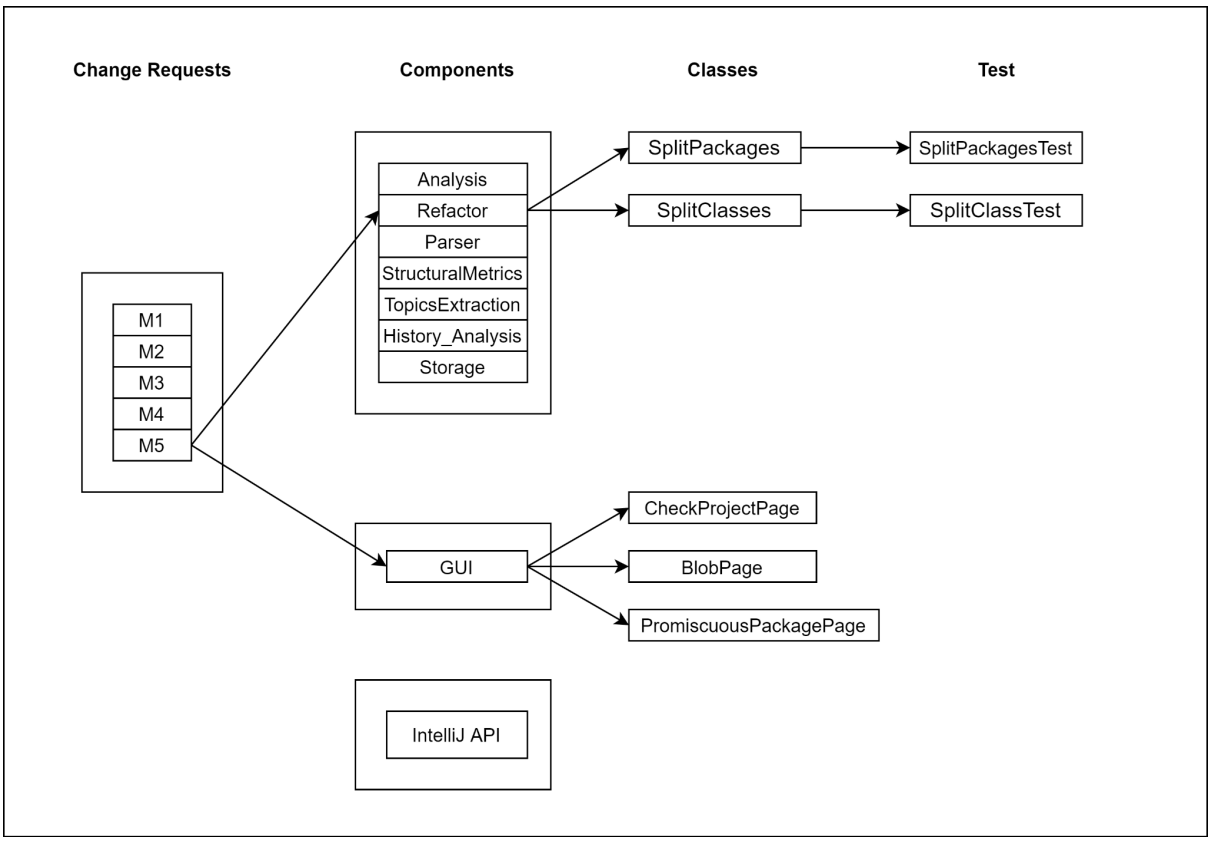


Diagramma di attività (M5)



Starting impact set (M5)

Nella costruzione della matrice delle dipendenze sono state prese in considerazione le classi con distanza 1. Anche per questa matrice sono state prese in considerazione solo le classi che potrebbero effettivamente impattare il sistema.

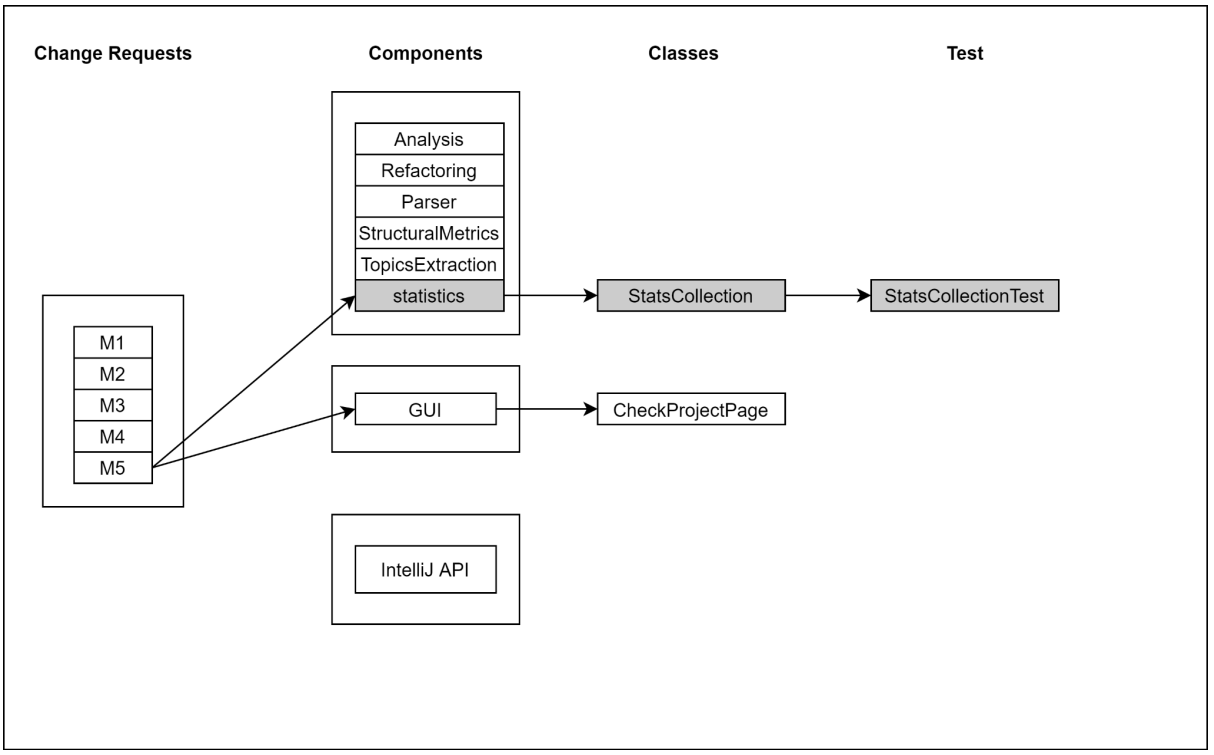
Matrice dipendenze M5	CheckProjectPage	BlobPage	PromiscuousPackagePage	SplitClasses	SplitPackages	CodeSmell	HistoryAnalysisStartup	ClassBean	MethodBean	PackageBean	CKMetrics
CheckProjectPage											
BlobPage											
PromiscuousPackagePage											
SplitClasses		1									
SplitPackages			1								
CodeSmell	1						2	2	2	2	
HistoryAnalysisStartup	1										
ClassBean	1	1			2						2
MethodBean	1				2						2
PackageBean	1		1		2						2
CKMetrics	1	1	1								

Candidate impact set (M5)

Il Candidate Impact Set è dunque formato dalle seguenti classi:

- CheckProjectPage
- SplitClasses
- SplitPackages
- CKMetrics
- CodeSmell

Dopo aver effettuato le modifiche, l'Actual Impact Set è mostrato nel grafico seguente:



Actual impact set (M5)

$$|FPIS| = |CIS \setminus AIS| = 5 - 1 = 4$$

$$Recall = \frac{|CIS \cap AIS|}{|AIS|} = 1/1 = 1$$

$$Precision = \frac{|CIS \cap AIS|}{|CIS|} = 1/5 = 0.2$$

$$Inclusiveness = 1$$

La *precision* molto bassa è dovuta ad una sovrastima del Candidate Impact Set. Tuttavia la modifica è stata effettuata con successo e senza problemi grazie al valore di *recall* pari a 1.

3.7 Tabella di riepilogo

Dalla tabella si nota che il valore della *precision* è risultato essere basso per ogni change request. Da ciò si può dedurre che la strategia adottata per l'individuazione del *candidate impact set* non è stata corretta in quanto ha in tutti i casi portato a una sovrastima dell'impatto effettivo. Le modifiche sono tuttavia state effettuate con successo e senza problemi, grazie al valore di *recall* pari a 1.

Change request	**|FPIS|**	**Recall**	**Precision**	**Adequacy**
M1	3	1	0.4	1
M2	2	1	0.5	1
M3	4	1	0.2	1
M4	4	1	0.2	1
M5	4	1	0.2	1

4 Piano di Test

Questo capitolo descrive l'approccio al processo di testing per il tool cASpER e definisce le regole su cui saranno basati i test, inclusi quelli già presenti nel progetto; inoltre sono descritti:

- il processo per stabilire e costruire dei test validi e come eseguirli;
- gli elementi da testare e quelli da non testare;
- le regole di decisione da usare per stabilire se una caratteristica software supera o meno il test;
- gli altri documenti previsti dal Piano di Test
- le attività eseguite per il testing;
- le responsabilità e i ruoli.

4.1 Master Test Plan

Lo scopo principale del piano di testing è quello di garantire che il funzionamento di cASpER sia quello stabilito dai requisiti e che eventuali fault possano essere individuati e corretti in future revisioni del tool.

4.1.1 Test Items

Il tool presenta già un insieme di test sviluppati in precedenza, i quali riguardano:

- l'analisi del codice, e quindi la detection dei code smell;
- il refactoring del codice.

In aggiunta a questi test, viene costruita una test suite per coprire altri aspetti funzionali del codice, e in particolare quelli critici che permettono il corretto utilizzo del tool. Alcuni di questi aspetti possono essere individuati banalmente nella specifica dei requisiti o durante il normale utilizzo del tool.

4.1.2 Funzionalità da testare

Verranno testate le funzionalità considerate critiche dal punto di vista dell'utente: in questo scenario l'utente deve poter avviare il tool, comprendere gli eventuali problemi che si verificano attraverso schermate dedicate, e usufruire delle funzionalità di analisi del codice e refactoring messe a disposizione da cASpER.
Per ogni funzionalità viene indicata una categoria di rischio (H - High, M - Medium, L - Low).

Num.	Funzionalità	Livello di rischio
1	Classe affetta da Blob smell	H
2	Classe affetta da Misplaced Class smell	H
3	Metodo affetto da Feature Envy smell	H
4	Pacchetto affetto da Promiscuous Package smell	H
5	Controllo della similarità tra entità	M
6	Controllo delle funzionalità di estrazione statistiche	M

4.1.3 Funzionalità da non testare

Funzionalità con livello di rischio idealmente più basso di tutte e tre le categorie indicate precedentemente non verranno testate. Inoltre non saranno testate le funzionalità che sono presenti nel codice il cui funzionamento non è previsto per l'attuale versione del tool: in particolare si tratta dell'analisi e del refactoring di tre code smell aggiuntivi:

- Divergent Change
- Shotgun Surgery
- Parallel Inheritance

4.1.4 Approccio

La strategia da attuare per il testing di cASpER è stata divisa in due livelli:

- Unit, Integration e System Testing
- Regression Testing

Secondo questo approccio viene verificato il comportamento dei moduli unitari del sistema prima ancora che questi possano interagire con altre componenti. Segue una fase di testing in cui viene verificato che il comportamento del sistema nel suo insieme soddisfa i requisiti. Infine viene effettuato testing di regressione a causa delle modifiche che devono essere effettuate sul sistema e che potrebbero impattare altre componenti dello stesso.

Il testing effettuato sarà di tipo funzionale per quanto concerne le funzionalità core del sistema, e di tipo strutturale per le parti del sistema che non interessano direttamente analisi del codice e refactoring, ovvero servizi di base e aggiuntivi.

4.1.4.1 Testing funzionale

Il testing funzionale verrà effettuato utilizzando il tool jUnit nel formato compatibile con IntelliJ IDEA e la libreria Mockito installata da Gradle, per creare gli stub necessari alla corretta esecuzione dei test. L'utilizzo base non prevede che si verifichino particolari condizioni nell'ambiente di sviluppo.

4.1.4.2 Testing strutturale

Per il testing strutturale verranno utilizzate tecniche riguardanti analisi statica/dinamica del codice. Il sistema sarà in esecuzione su una istanza virtualizzata di IntelliJ IDEA su cui è installato il plugin cASpER.

4.1.5 Criteri di pass/fail

In base ad ogni specifico livello di testing, vengono stabiliti dei criteri diversi di superamento o fallimento dei test. In particolare per ogni test case viene dichiarato l'output atteso e quello effettivo, in modo da poter chiaramente distinguere un comportamento corretto da uno inaspettato.

4.1.6 Test Tasks

Ogni test viene identificato per poter avere una visione sistematica del processo di testing. Vengono definite le configurazioni da utilizzare e i requisiti da rispettare.

4.1.7 Environmental needs

Per eseguire il tool è necessaria una versione di IntelliJ IDEA (Ultimate o Community Edition) inferiore alla 2020, si preferisce la 2019.3.5. Gradle deve essere installato e può essere aggiornato senza problemi. La versione del Java Development Kit deve essere inferiore alla 11, e si preferisce la 1.8.x.

Per eseguire il tool e quindi verificarne il corretto funzionamento è necessario creare una nuova configurazione di esecuzione in IntelliJ IDEA, la quale esegue il comando `:runIde` e ha come argomenti a linea di comando `-stacktrace`.

Per eseguire il test occorre avviare il metodo `main` nella classe `TestRunner` oppure creare una configurazione di test su IntelliJ in modo da poter eseguire la suite presente nella classe `JUnitTestSuite`.

4.1.8 Responsabilità e ruoli

Il team di sviluppo si occuperà di effettuare il testing del sistema. I ruoli e le responsabilità sono indicate nella tabella seguente.

Ruolo	**Responsabilità**
Test Manager	Organizzazione del processo di testing Acquisizione delle risorse necessarie Reporting di comportamenti anomali Valutazione dell'efficacia dei test
Test Designer	Definizione dell'approccio al testing Definizione dell'architettura del testing Verifica delle tecniche di test Definizione delle entità da testare Implementazione dei test
Tester	Esecuzione dei test Report dei test Risoluzione di test falliti

4.1.9 Staffing and Training needs

Requisito necessario allo sviluppo di un corretto piano di test è aver seguito e compreso il corso di Ingegneria, Gestione ed Evoluzione del Software nell'Anno Accademico corrente, o averlo seguito negli anni precedenti e averne ancora memoria. Una buona preparazione e collaborazione tra colleghi contribuisce alla produzione di deliverables di qualità e ad una corretta stima degli effort necessari allo svolgimento del progetto del corso.

4.1.10 Schedule

Il processo di testing segue quello di reverse engineering effettuato sul sistema. Prima di implementare le modifiche designate nel documento di Impact Analysis, viene effettuato il test del sistema che come detto precedentemente comprende l'esecuzione dei test già presenti e la creazione di una test suite aggiuntiva. A seguito di ciò verranno apportate le modifiche al sistema e dopo un test di regressione verranno individuati, creati ed eseguiti nuovi casi di test.

4.1.11 Rischi

I rischi considerati sono indicati nella tabella seguente.

Categoria di rischio	Area di rischio	Misura di mitigazione del rischio
Contagio da COVID-19	Nel caso di contagio da malattia da COVID-19 in maniera non asintomatica di un componente del team, la data di fine progetto potrebbe slittare.	
	Nel caso di contagio da malattia da COVID-19 in maniera non asintomatica di un componente del team, eventuali artefatti non condivisi potrebbero rallentare o inibire lo sviluppo di determinate parti del progetto.	Restare allineati in tutte le fasi del progetto e tenersi in contatto.
Documentazione non adeguata	Rallentamento dello sviluppo per il reverse engineering	
	Rallentamento nella costruzione di un ambiente efficace per l'esecuzione del sistema (configurazione di IntelliJ).	Salvare la configurazione dell'ambiente.
Difficoltà nella creazione di casi di test adeguati		Collaborazione e aumento dell'effort per la risoluzione del problema.

4.2 Unit e Integration Test Plan

Questo paragrafo mette in risalto le scelte effettuate e l'approccio seguito per il processo di testing di unità e di integrazione di cASpER. Nella sezione 4.2.1 viene mostrata la struttura della suite di test, indicando i test già presenti nel sistema, i test addizionali inseriti per le funzionalità più a basso rischio, e quelli aggiunti dopo aver effettuato il test di regressione.

L’approccio ha visto l’esecuzione e il controllo dei test prima di effettuare modifiche al sistema. Successivamente all’implementazione delle modifiche, è stata selezionata la suite di test da utilizzare per il test di regressione. I risultati sono indicati nella sezione di Test Report.

4.2.1 Unit e Integration Test Design

I test descritti in questa sezione sono stati scritti utilizzando il framework JUnit e la libreria Mockito per Java. Gli oracoli sono stati definiti osservando i requisiti del tool e il comportamento del codice. I test vengono considerati passati quando JUnit non rileva errori.
La versione attuale di cASpER presenta una suite di test composta dalle seguenti caratteristiche:

Nome classe	Pacchetto	Metodi di test
StructuralBlobStrategyTest	it.unisa.casper.analysis.code_smell_detection.blob	5
TextualBlobStrategyTest	it.unisa.casper.analysis.code_smell_detection.blob	4
StructuralFeatureEnvyStrategyTest	it.unisa.casper.analysis.code_smell_detection.feature_envy	4
TextualFeatureEnvyStrategyTest	it.unisa.casper.analysis.code_smell_detection.feature_envy	4
StructuralMisplacedClassStrategyTest	it.unisa.casper.analysis.code_smell_detection.misplaced_class	4
TextualMisplacedClassStrategyTest	it.unisa.casper.analysis.code_smell_detection.misplaced_class	4
StructuralPromiscuousPackageStrategyTest	it.unisa.casper.analysis.code_smell_detection.promiscuous_package	4
TextualPromiscuousPackageStrategyTest	it.unisa.casper.analysis.code_smell_detection.promiscuous_package	4
BeanDetectionTest	it.unisa.casper.analysis.code_smell_detection	1
SplitClassesTest	it.unisa.casper.refactor	2
SplitPackagesTest	it.unisa.casper.refactor	2

Successivamente sono state individuate nuove entità da testare, mostrate di seguito:

Nome classe	Pacchetto	Metodi di test
BlobCodeSmellTest	it.unisa.casper.analysis.code_smell	2
FeatureEnvyCodeSmellTest	it.unisa.casper.analysis.code_smell	2
MisplacedClassCodeSmellTest	it.unisa.casper.analysis.code_smell	2
PromiscuousPackageCodeSmell Test	it.unisa.casper.analysis.code_smell	2
BeanComparatorTest	it.unisa.casper.analysis.code_smell_ detection.comparator	3
CosineSimilarityTest	it.unisa.casper.analysis.code_smell_ detection.similarityComputation	2
SmellynessMetricTest	it.unisa.casper.analysis.code_smell_ detection.smellynessMetricProcessi ng	1

Infine, sono stati creati i test per le funzionalità aggiunte dalle CR mostrate nel primo capitolo:

Nome classe	Pacchetto	Metodi di test
GlobalTimerTest	it.unisa.casper.statistics	2
StatsCollectionTest	it.unisa.casper.statistics	1
StatsExtracterTest	it.unisa.casper.statistics	3

4.2.2 Unit e Integration Test Case

Le caratteristiche da testare individuate nel paragrafo precedente sono ora specificate nei test case riportati di seguito.

Test ID	1	**Pacchetto**	it.unisa.casper.analysis.code_smell
Classe	BlobCodeSmellTest	**Prerequisiti**	stub delle classi ClassBean, CodeSmellDetectionStrategy

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**

1	affectsTrue()	affects()	true	**pass**	Crea un'istanza di classBean affetto da smell (tramite isSmelly() e verifica che il metodo affects restituisce true
2	affectsFalse()	affects()	false	**pass**	verifica che il metodo affects restituisce false su un'istanza arbitraria

Test ID	2	**Pacchetto**	it.unisa.casper.analysis.code_smell
Classe	FeatureEnvyCodeSmellTest	**Prerequisiti**	stub delle classi CodeSmellDetectionStrategy, MethodBean

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**
1	affectsTrue()	affects()	true	**pass**	Crea un'istanza di MethodBean affetto da smell (tramite isSmelly() e verifica che il metodo affects restituisce true
2	affectsFalse()	affects()	false	**pass**	verifica che il metodo affects restituisce false su un'istanza arbitraria

Test ID	3	**Pacchetto**	it.unisa.casper.analysis.code_smell
Classe	MisplacedClassCodeSmellTest	**Prerequisiti**	stub delle classi CodeSmellDetectionStrategy, ClassBean

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**

1	affectsTrue()	affects()	true	**pass**	Crea un'istanza di ClassBean affetto da smell (tramite isSmelly() e verifica che il metodo affects restituisce true
2	affectsFalse()	affects()	false	**pass**	verifica che il metodo affects restituisce false su un'istanza arbitraria

Test ID	4	**Pacchetto**	it.unisa.casper.analysis.code_smell
Classe	PromiscuousPackageCode SmellTest	**Prerequisiti**	stub delle classi CodeSmellDetectionStrategy, PackageBean

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**
1	affectsTrue()	affects()	true	**pass**	Crea un'istanza di PackageBean affetto da smell (tramite isSmelly() e verifica che il metodo affects restituisce true
2	affectsFalse()	affects()	false	**pass**	verifica che il metodo affects restituisce false su un'istanza arbitraria

Test ID	5	**Pacchetto**	it.unisa.casper.analysis.code_smell
Classe	PromiscuousPackageCode SmellTest	**Prerequisiti**	stub delle classi CodeSmellDetectionStrategy, PackageBean

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**

1	affectsTrue()	affects()	true	pass	Crea un'istanza di PackageBean affetto da smell (tramite isSmelly() e verifica che il metodo affects restituisce true
2	affectsFalse()	affects()	false	pass	verifica che il metodo affects restituisce false su un'istanza arbitraria

Test ID	6	**Pacchetto**	it.unisa.casper.analysis.code_smell_detection.blob
Classe	StructuralBlobStrategyTest	**Prerequisiti**	stub creati con MethodBeanList, MethodBean.Builder, ClassBean.Builder, InstanceVariableBean

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**
1	isSmellyTrue()	isAffected()	true	pass	Crea un'istanza di ClassBean affetta da Blob smell e verifica secondo la strategia strutturale che venga riconosciuta come affetta da smell.
2	isSmellyNearThreshold()	isAffected()	true	pass	Crea un'istanza di ClassBean affetta da Blob smell con un valore vicino alla soglia stabilita e verifica secondo la strategia strutturale che venga riconosciuta come tale.
3	isSmellyMinThreshold()	isAffected()	false	pass	Crea un'istanza di ClassBean affetta da Blob smell con un valore sotto la soglia minima e verifica secondo la strategia strutturale che non venga riconosciuta come tale.
4	isSmellyTrueControl()	isAffected()	true	pass	Crea un'istanza di ClassBean affetta da

					Blob smell e verifica secondo la strategia strutturale che venga riconosciuta come affetta da smell.
5	isSmellyFalse()	isAffected()	false	**pass**	Crea un'istanza di ClassBean non affetta da Blob smell e verifica secondo la strategia strutturale che venga riconosciuta come non affetta da smell.

Test ID	7	**Pacchetto**	it.unisa.casper.analysis.code_smell_detection.blob
Classe	TextualBlobStrategyTest	**Prerequisiti**	stub creati con MethodBeanList, MethodBean.Builder, ClassBean.Builder, InstanceVariableBean

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**
1	isSmellyTrue()	isAffected()	true	**pass**	Crea un'istanza di ClassBean affetta da Blob smell e verifica secondo la strategia testuale che venga riconosciuta come affetta da smell.
2	isSmellyNearThreshold()	isAffected()	true	**pass**	Crea un'istanza di ClassBean affetta da Blob smell con un valore vicino alla soglia stabilita e verifica secondo la strategia testuale che venga riconosciuta come tale.
3	isSmellyMinThreshold()	isAffected()	false	**pass**	Crea un'istanza di ClassBean affetta da Blob smell con un valore sotto la soglia minima e verifica secondo la strategia testuale che non venga riconosciuta come tale.

4	isSmellyTrueControl()	isAffected()	true	**pass**	Crea un'istanza di ClassBean affetta da Blob smell e verifica secondo la strategia strutturale che venga riconosciuta come affetta da smell.
5	isSmellyFalse()	isAffected()	false	**pass**	Crea un'istanza di ClassBean non affetta da Blob smell e verifica secondo la strategia testuale che venga riconosciuta come non affetta da smell.

Test ID	8	**Pacchetto**	it.unisa.casper.analysis.code_smell_detection.feature_envy
Classe	StructuralFeatureEnvyStrategyTest	**Prerequisiti**	stub creati con MethodBeanList, MethodBean.Builder, ClassBean.Builder, ClassBeanList, InstanceVariableBean, PackageBean.Builder

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**
1	isSmellyTrue()	isAffected()	true	**pass**	Crea un'istanza di MethodBean affetta da Feature Envy smell e verifica secondo la strategia strutturale che venga riconosciuta come affetta da smell.
2	isSmellyNearThreshold()	isAffected()	true	**pass**	Crea un'istanza di MethodBean affetta da Feature Envy smell con un valore vicino alla soglia stabilita e verifica secondo la strategia strutturale che venga riconosciuta come tale.
3	isSmellyMinThre	isAffected()	false	**pass**	Crea un'istanza di

	shold()				MethodBean affetta da Feature Envy smell con un valore sotto la soglia minima e verifica secondo la strategia strutturale che non venga riconosciuta come tale.
4	isSmellyFalse()	isAffected()	false	**pass**	Crea un'istanza di MethodBean non affetta da Feature Envy smell e verifica secondo la strategia strutturale che venga riconosciuta come non affetta da smell.

Test ID	9	**Pacchetto**	it.unisa.casper.analysis.code_smell_detection.feature_envy
Classe	TextualFeatureEnvyStrategyTest	**Prerequisiti**	stub creati con MethodBeanList, MethodBean.Builder, ClassBean.Builder, ClassBeanList, InstanceVariableBean, PackageBean.Builder

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**
1	isSmellyTrue()	isAffected()	true	**pass**	Crea un'istanza di MethodBean affetta da Feature Envy smell e verifica secondo la strategia testuale che venga riconosciuta come affetta da smell.
2	isSmellyNearThreshold()	isAffected()	true	**pass**	Crea un'istanza di MethodBean affetta da Feature Envy smell con un valore vicino alla soglia stabilita e verifica secondo la strategia testuale che venga riconosciuta come tale.

3	isSmellyMinThreshold()	isAffected()	false	**pass**	Crea un'istanza di MethodBean affetta da Feature Envy smell con un valore sotto la soglia minima e verifica secondo la strategia testuale che non venga riconosciuta come tale.
4	isSmellyFalse()	isAffected()	false	**pass**	Crea un'istanza di MethodBean non affetta da Feture Envy smell e verifica secondo la strategia testuale che venga riconosciuta come non affetta da smell.

Test ID	10	**Pacchetto**	it.unisa.casper.analysis.code_smell_detection.misplaced_class
Classe	StructuralMisplacedClassStrategyTest	**Prerequisiti**	stub creati con MethodBeanList, MethodBean.Builder, ClassBean.Builder, ClassBeanList, InstanceVariableBean, PackageBean.Builder

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**
1	isSmellyTrue()	isAffected()	true	**pass**	Crea un'istanza di ClassBean affetta da Misplaced Class smell e verifica secondo la strategia strutturale che venga riconosciuta come affetta da smell.
2	isSmellyNearThreshold()	isAffected()	true	**pass**	Crea un'istanza di ClassBean affetta da Misplaced Class smell con un valore vicino alla soglia stabilita e verifica secondo la strategia strutturale che venga riconosciuta come tale.
3	isSmellyMinThres	isAffected()	false	**pass**	Crea un'istanza di

	hold()				ClassBean affetta da Misplaced Class smell con un valore sotto la soglia minima e verifica secondo la strategia strutturale che non venga riconosciuta come tale.
4	isSmellyFalse()	isAffected()	false	**pass**	Crea un'istanza di ClassBean non affetta da Misplaced Class smell e verifica secondo la strategia strutturale che venga riconosciuta come non affetta da smell.

Test ID	11	**Pacchetto**	it.unisa.casper.analysis.code_smell_detection.misplaced_class
Classe	TextuaMisplacedClassStrategyTest	**Prerequisiti**	stub creati con MethodBeanList, MethodBean.Builder, ClassBean.Builder, ClassBeanList, InstanceVariableBean, PackageBean.Builder

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**
1	isSmellyTrue()	isAffected()	true	**pass**	Crea un'istanza di ClassBean affetta da Misplaced Class smell e verifica secondo la strategia testuale che venga riconosciuta come affetta da smell.
2	isSmellyNearThreshold()	isAffected()	true	**pass**	Crea un'istanza di ClassBean affetta da Misplaced Class smell con un valore vicino alla soglia stabilita e verifica secondo la strategia testuale che venga riconosciuta come tale.
3	isSmellyMinThreshold()	isAffected()	false	**fail**	Crea un'istanza di ClassBean affetta da Misplaced Class smell con un valore sotto la

					soglia minima e verifica secondo la strategia testuale che non venga riconosciuta come tale.
4	isSmellyFalse()	isAffected()	false	**pass**	Crea un'istanza di ClassBean non affetta da Misplaced Class smell e verifica secondo la strategia testuale che venga riconosciuta come non affetta da smell.

Test ID	12	**Pacchetto**	it.unisa.casper.analysis.code_smell_detection.promiscuous_package
Classe	StructuralPromiscuousPackageStrategyTest	**Prerequisiti**	stub creati con MethodBeanList, MethodBean.Builder, ClassBean.Builder, ClassBeanList, InstanceVariableBean, PackageBean.Builder

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**
1	isSmellyTrue()	isAffected()	true	**pass**	Crea un'istanza di PackageBean affetta da Promiscuous Package smell e verifica secondo la strategia strutturale che venga riconosciuta come affetta da smell.
2	isSmellyNearThreshold()	isAffected()	true	**pass**	Crea un'istanza di PackageBean affetta da Promiscuous Package smell con un valore vicino alla soglia stabilita e verifica secondo la strategia strutturale che venga riconosciuta come tale.
3	isSmellyMinThreshold()	isAffected()	false	**pass**	Crea un'istanza di PackageBean affetta da Promiscuous Package smell con un valore sotto la soglia minima e verifica secondo la strategia strutturale che

					non venga riconosciuta come tale.
4	isSmellyFalse()	isAffected()	false	pass	Crea un'istanza di PackageBean non affetta da Promiscuous Package smell e verifica secondo la strategia strutturale che venga riconosciuta come non affetta da smell.

Test ID	13	**Pacchetto**	it.unisa.casper.analysis.code_smell_detection.promiscuous_package
Classe	TextualPromiscuousPackageStrategyTest	**Prerequisiti**	stub creati con MethodBeanList, MethodBean.Builder, ClassBean.Builder, ClassBeanList, InstanceVariableBean, PackageBean.Builder

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**
1	isSmellyTrue()	isAffected()	true	pass	Crea un'istanza di PackageBean affetta da Promiscuous Package smell e verifica secondo la strategia testuale che venga riconosciuta come affetta da smell.
2	isSmellyNearThreshold()	isAffected()	true	pass	Crea un'istanza di PackageBean affetta da Promiscuous Package smell con un valore vicino alla soglia stabilita e verifica secondo la strategia testuale che venga riconosciuta come tale.
3	isSmellyMinThreshold()	isAffected()	false	pass	Crea un'istanza di PackageBean affetta da Promiscuous Package smell con un valore sotto la soglia minima e verifica secondo la strategia testuale che non venga riconosciuta come tale.
4	isSmellyFalse()	isAffected()	false	pass	Crea un'istanza di PackageBean non affetta da Promiscuous Package smell e verifica secondo la strategia testuale che venga riconosciuta come non affetta da smell.

Test ID	14	**Pacchetto**	it.unisa.casper.analysis.code_smell_detection.similarityComputation
Classe	CosineSimilarityTest	**Prerequisiti**	stub della classe CosineSimilarity

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**
1	computeSimilarityFileTest()	computeSimilarity()	true	**pass**	Invoca il metodo dello stub e controlla che il file definito dalla classe esista.
2	readFileTest()	CosineSimilarity.readFile()	equality check	**pass**	Crea un file temporaneo e controlla che il metodo legga correttamente il file.

Test ID	15	**Pacchetto**	it.unisa.casper.analysis.code_smell_detection.smellynessMetricProcessing
Classe	SmellynessMetricTest	**Prerequisiti**	

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**
1	computeSmellynessExceptionTest()	computeSmellyness()	Exception throw	**pass**	Invoca il metodo passando null come parametro e controlla che sia lanciata un'eccezione.

Test ID	16	**Pacchetto**	it.unisa.casper.analysis.code_smell_detection

Classe	BeanDetectionTest	**Prerequisiti**	stub di metodi e classi creati con MethodBean.Builder e ClassBean.Builder

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**
1	isBean()	BeanDetection.detection()	true	**pass**	

Test ID	17	**Pacchetto**	it.unisa.casper.analysis.refactor
Classe	SplitClassTest	**Prerequisiti**	stub di metodi e classi e pacchetti creati con MethodBean.Builder, ClassBean.Builder e PackageBean.Builder

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**
1	splitTrue()	split()	true	**pass**	Controlla che il metodo invocato divida correttamente le classi affette da smell
2	splitFalse()	split()	true	**pass**	Controlla che il metodo invocato non divida le classi in quanto non affette da smell.

Test ID	18	**Pacchetto**	it.unisa.casper.analysis.refactor
Classe	SplitPackagesTest	**Prerequisiti**	stub di metodi e classi e pacchetti creati con MethodBean.Builder, ClassBean.Builder e PackageBean.Builder

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**
1	splitTrue()	split()	true	**pass**	Controlla che il metodo

					invocato divida correttamente i pacchetti affetti da smell
2	splitFalse()	split()	true	**pass**	Controlla che il metodo invocato non divida i pacchetti in quanto non affetti da smell.

Test ID	19	**Pacchetto**	it.unisa.casper.analysis.code_smell_detection.comparator
Classe	BeanComparatorTest	**Prerequisiti**	stub di ComparableBean

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**
1	compareLessThan()	compare()	Equality check	**pass**	Controlla che il metodo invocato confronti correttamente i valori passati in input dei due Bean che sono in relazione di minore/maggiore.
2	compareEquals()	compare()	Equality check	**pass**	Controlla che il metodo invocato confronti correttamente i valori passati in input dei due Bean che sono in relazione di uguaglianza.
3	compareMoreThan()	compare()	Equality check	**pass**	Controlla che il metodo invocato confronti correttamente i valori passati in input dei due Bean che sono in relazione di minore/maggiore.

Test ID	20	**Pacchetto**	it.unisa.casper.analysis.code_smell_detection.smellynessMetricProcessing

Classe	SmellynessMetricTest	**Prerequisiti**	stub di CosineSimilarity

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**
1	computeSmellynessExceptionTest()	computeSmellyness()	Exception thrown	**pass**	Controlla che il metodo invocato lanci correttamente un'eccezione in caso di argomento nullo.

Test ID	21	**Pacchetto**	it.unisa.casper.analysis.statistics
Classe	GlobalTimerTest	**Prerequisiti**	stub della classe GlobalTimer

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**
1	stopViewTimeTest()	stopViewTime()	verify invocation	**pass**	Controlla che il metodo sia invocato correttamente.
2	stopExecutionTimeTest()	stopExecutionTime()	verify invocation	**pass**	Controlla che il metodo sia invocato correttamente.

Test ID	22	**Pacchetto**	it.unisa.casper.analysis.statistics
Classe	StatsCollectionTest	**Prerequisiti**	stub della classe StatsCollection

Test Case n°	**Nome**	**Metodo da testare**	**Output atteso**	**Risultato**	**Commenti**
1	createCSVTest()	createCSV()	Exception throw	**pass**	Controlla che il metodo invocato lanci correttamente un'eccezione.

Test ID	23	Pacchetto	it.unisa.casper.analysis.statistics
Classe	StatsExtracterTest	Prerequisiti	stub della classe StatsExtracterTest

Test Case n°	Nome	Metodo da testare	Output atteso	Risultato	Commenti
1	readStatisticsFileCheckTest()	readStatistics()	true	pass	Controlla che il file che viene creato dal metodo invocato esista.
2	readStatisticsNegativeIndexTest()	readStatistics()	Exception throw	pass	Controlla che il metodo invocato lanci correttamente un'eccezione.
3	readStatisticsPositiveIndexTest()	readStatistics()	Equality check	pass	Controlla che il file che viene creato dal metodo invocato esista.

4.2.3 Unit e Integration Test Report

In questa sezione vengono riportate le statistiche relative all'esecuzione dei test indicati in precedenza ed eventuali errori che si sono verificati.
Il numero totale di casi di test presente nel sistema è 58 e comprende test di unità e test di integrazione. Per rendere possibile l'esecuzione in blocco dei test è necessaria la creazione di una test suite tramite IntelliJ.

Eseguendo la test suite prima di effettuare le modifiche al sistema si ottiene il seguente risultato:

Test passati con successo	51/52	Test falliti	1/52

Il test fallito è identificato dal **Test Case ID** 11, riguardante la classe `TextualMisplacedClassStrategyTest`. Il metodo in questione è `isSmellyMinThreshold()`, il cui comportamento atteso sarebbe quello di verificare secondo la strategia testuale che un'istanza di ClassBean creata e affetta da Misplaced Class smell con un valore sotto la soglia minima non venga riconosciuta come tale. Osservando la

natura del test e la history del codice sorgente, si evince che il test passava con successo in una versione precedente del tool.

4.3 System Test Plan

Questo paragrafo mette in risalto le scelte effettuate e l'approccio seguito per il processo di testing di sistema di cASpER. Nella sezione 4.3.1 viene mostrata la struttura dei test individuati e l'esecuzione del test di regressione.

L'approccio ha visto l'esecuzione e il controllo dei test prima di effettuare modifiche al sistema. Successivamente all'implementazione delle modifiche, sono stati individuati nuovi test per le funzionalità aggiunte. I risultati sono indicati nella sezione di Test Report.

4.3.1 System Test Design

I test descritti in questa sezione sono stati individuati osservando i requisiti del tool e comprendendo il corretto funzionamento dello stesso. Un test è considerato passato quando il comportamento del sistema è uguale a quello definito dall'oracolo.

Le funzionalità che vengono testate in questo livello di test riguardano principalmente il corretto funzionamento dell'interfaccia grafica e di alcune funzionalità di base. Le caratteristiche individuate per il testing sono mostrate di seguito:

Test	Scenari interni	Commenti
Avvio di cASpER	2	
Visualizzazione di un Blob smell	2	L'interfaccia che viene costruita dal tool è la stessa per ogni smell, quindi verrà testata solo quella riguardante il Blob.
Refactoring di un Blob smell	2	L'interfaccia che viene costruita dal tool è la stessa per ogni smell, quindi verrà testata solo quella riguardante il Blob.
Visualizzazione statistiche	3	
Esportazione CSV	1	
Visualizzazione delle statistiche degli ultimi 10 utilizzi del tool	1	La logica è la stessa per ogni intervallo, quindi verrà testata solo questa particolare condizione.

4.3.2 System Test Case

Le caratteristiche da testare individuate nel paragrafo precedente sono ora specificate nei test case riportati di seguito.

Test Scenario ID	1	**Test Case ID**	Avvio-1
Descrizione	Avvio di cASpER - positivo	**Priorità**	Alta
Prerequisiti	Installazione del plugin cASpER	**Postrequisiti**	N/A

Step di esecuzione					
Step n°	**Azione**	**Input**	**Output atteso**	**Risultato**	**Commenti**
1	Tools > cASpER - Analyze Project > Run plug-in	Un qualsiasi progetto ben formato	Schermata principale di cASpER	**pass**	Se il progetto è grosso ci mette un po’

Test Scenario ID	1	**Test Case ID**	Avvio-2
Descrizione	Avvio di cASpER - negativo	**Priorità**	Alta
Prerequisiti	Installazione del plugin cASpER	**Postrequisiti**	N/A

Step di esecuzione					
Step n°	**Azione**	**Input**	**Output atteso**	**Risultato**	**Commenti**
1	Tools > cASpER - Analyze Project > Run plug-in	Un progetto malformato, ad esempio con cartelle mancanti	“Sorry, an error has occurred. Please try again or contact support”	**pass**	

Test Scenario ID	2	**Test Case ID**	VisualizzazioneSmell-1
Descrizione	Visualizzazione di un blob smell - negativo	**Priorità**	Media
Prerequisiti	Installazione del plugin cASpER	**Postrequisiti**	N/A

Step di esecuzione					
Step n°	**Azione**	**Input**	**Output atteso**	**Risultato**	**Commenti**
1	Tools > cASpER - Analyze Project > Run plug-in	Un progetto contenente una classa affetta da blob smell	Homepage di cASpER	**pass**	
2	Clicca INSPECT	N/A	“Warning - select an item”	**pass**	Una volta selezionata una riga della tabella questo test case non può essere eseguito a meno che non si riavvia il tool.

Test Scenario ID	2	**Test Case ID**	VisualizzazioneSmell-2
Descrizione	Visualizzazione di un blob smell - positivo	**Priorità**	Media
Prerequisiti	Installazione del plugin cASpER	**Postrequisiti**	N/A

Step di esecuzione					
Step n°	**Azione**	**Input**	**Output atteso**	**Risultato**	**Commenti**
1	Tools > cASpER - Analyze Project > Run plug-in	Un progetto contenente una classa affetta da blob smell	Homepage di cASpER	**pass**	
2	Seleziona la riga della tabella in cui è indicato il blob smell	N/A	Il codice in esame viene mostrato nella	**pass**	

			TextArea a destra		
3	Clicca INSPECT	N/A	Schermata di Blob Class Analysis	**pass**	

Test Scenario ID	3	**Test Case ID**	RefactoringBlobSmell-1
Descrizione	Refactoring di un blob smell - positivo	**Priorità**	Alta
Prerequisiti	Installazione del plugin cASpER	**Postrequisiti**	N/A

Step di esecuzione					
Step n°	**Azione**	**Input**	**Output atteso**	**Risultato**	**Commenti**
1	Tools > cASpER - Analyze Project > Run plug-in	Un progetto contenente una classa affetta da blob smell	Homepage di cASpER	**pass**	
2	Seleziona la riga della tabella in cui è indicato il blob smell	N/A	Il codice in esame viene mostrato nella TextArea a destra	**pass**	
3	Clicca INSPECT	N/A	Schermata di Blob Class Analysis	**pass**	
4	Clicca FIND SOLUTION	N/A	Schermata di Blob Refactoring	**pass**	
5	Clicca REFACTOR	Selezione di default del tool	“Blob Corrected, check new classes generated name”	**pass**	

Test Scenario ID	3	**Test Case ID**	RefactoringBlobSmell-2
Descrizione	Refactoring di un blob smell - negativo	**Priorità**	Alta

Prerequisiti	Installazione del plugin cASpER	**Postrequisiti**	N/A

Step di esecuzione					
Step n°	**Azione**	**Input**	**Output atteso**	**Risultato**	**Commenti**
1	Tools > cASpER - Analyze Project > Run plug-in	Un progetto contenente una classa affetta da blob smell - si assume che la classe non può essere refattorizzata senza introdurre nuovi smell	Homepage di cASpER	**pass**	
2	Seleziona la riga della tabella in cui è indicato il blob smell	N/A	Il codice in esame viene mostrato nella TextArea a destra	**pass**	
3	Clicca INSPECT	N/A	Schermata di Blob Class Analysis	**pass**	
4	Clicca FIND SOLUTION	N/A	“Error during creation of solution”	**pass**	

Test Scenario ID	4	**Test Case ID**	VisualizzazioneStatistiche-1
Descrizione	Visualizzazione statistiche - negativo	**Priorità**	Bassa
Prerequisiti	File delle statistiche malformato	**Postrequisiti**	Controllare che il file delle statistiche sia ben formato

Step di esecuzione					
Step n°	**Azione**	**Input**	**Output atteso**	**Risultato**	**Commenti**
1	Tools > cASpER - Analyze Project > View statistics	N/A	“Error fetching statistics”	**pass**	

Test Scenario ID	4	**Test Case ID**	VisualizzazioneStatistiche-2
Descrizione	Visualizzazione statistiche - negativo	**Priorità**	Bassa
Prerequisiti	Non aver mai avviato il tool	**Postrequisiti**	Avviare il tool prima di controllare le statistiche

Step di esecuzione					
Step n°	**Azione**	**Input**	**Output atteso**	**Risultato**	**Commenti**
1	Tools > cASpER - Analyze Project > View statistics	N/A	“Error fetching statistics”, schermata delle statistiche vuota	**pass**	

Test Scenario ID	4	**Test Case ID**	VisualizzazioneStatistiche-3
Descrizione	Visualizzazione statistiche - positivo	**Priorità**	Bassa
Prerequisiti	Installazione del plugin cASpER	**Postrequisiti**	

Step di esecuzione					
Step n°	**Azione**	**Input**	**Output atteso**	**Risultato**	**Commenti**
1	Tools > cASpER - Analyze Project > View statistics	N/A	Schermata delle statistiche	**pass**	

Test Scenario ID	5	**Test Case ID**	EsportazioneCSV-1
Descrizione	Esportazione CSV	**Priorità**	Bassa
Prerequisiti	Installazione del plugin cASpER	**Postrequisiti**	

Step di esecuzione					
Step n°	**Azione**	**Input**	**Output atteso**	**Risultato**	**Commenti**

1	Tools > cASpER - Analyze Project > View statistics	N/A	Schermata delle statistiche	**pass**	
2	Clicca EXPORT .CSV	N/A	Apertura del file manager che punta al file delle statistiche	**pass**	Se il file manager non può visualizzare le cartelle nascoste, questo potrebbe essere l'unico metodo per accedere al file.

Test Scenario ID	6	**Test Case ID**	FiltroStatistiche-1
Descrizione	Visualizzazione delle statistiche degli ultimi 10 utilizzi del tool	**Priorità**	Bassa
Prerequisiti	Aver avviato cASpER almeno 10 volte	**Postrequisiti**	

Step di esecuzione					
Step n°	**Azione**	**Input**	**Output atteso**	**Risultato**	**Commenti**
1	Tools > cASpER - Analyze Project > View statistics	N/A	Schermata delle statistiche	**pass**	
2	Clicca sulla combobox	Last 10 usages	N/A	N/A	
	Clicca su APPLY	N/A	I grafici vengono aggiornati con i nuovi dati	**pass**	Se il tool è stato avviato meno di 10 volte, verranno visualizzate le statistiche di tutti gli utilizzi.

4.3.3 System Test Report

In questa sezione vengono riportate le statistiche relative all'esecuzione dei test indicati in precedenza ed eventuali errori che si sono verificati.
Il numero totale di casi di test individuati nella sezione precedente è 11, nei quali sono compresi i test che riguardano le funzionalità aggiunte. Il risultato dell'esecuzione è il seguente:

Test passati con successo	11/11	Test falliti	0/11

I test non mostrano comportamenti anomali e passano tutti con successo.

4.4 Test di regressione

Dopo aver effettuato le modifiche indicate nelle CR al sistema, viene eseguita nuovamente la suite di test per controllare che le modifiche non abbiano alterato il comportamento del sistema. A questi casi di test si aggiungono quelli identificati per le funzionalità aggiunte. Il risultato ottenuto è il seguente:

Test passati con successo	57/58	Test falliti	1/58

Il test fallito è ancora una volta quello mostrato in precedenza. I nuovi test passano correttamente e dunque è possibile concludere che il test di regressione è stato effettuato con successo.

4.5 Coverage report

In seguito ai test effettuati è stata calcolata la *code coverage*, metrica utilizzata per definire la quantità di codice eseguito durante i test automatici. Per calcolare la copertura è stato usato il tool **JaCoCo** integrato in IntelliJ IDEA.
La tabella seguente mostra la copertura per ogni pacchetto di classi, metodi, righe di codice e branch.

Package	Class %	Method %	Line %	Branch %
it.unisa.casper.analysis.code_smell_detection.promiscuous_package	100% (2/2)	100% (6/6)	88% (30/34)	100% (6/6)
it.unisa.casper.analysis.code_smell_detection.comparator	100% (1/1)	100% (1/1)	100% (6/6)	100% (4/4)
it.unisa.casper.analysis	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
it.unisa.casper.analysis.code_smell_detection.strategy	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
it.unisa.casper.refactor	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
it.unisa.casper.refactor.exceptions	0% (0/5)	0% (0/5)	0% (0/10)	100% (0/0)
it.unisa.casper.refactor.strategy	0% (0/1)	0% (0/2)	0% (0/5)	100% (0/0)

it.unisa.casper.storage	100% (0/0)	100% (0/0)	100% (0/0)	100% (0/0)
it.unisa.casper.analysis.code_smell_detection.smellynessMetricProcessing	100% (1/1)	100% (1/1)	96% (26/27)	92% (13/14)
it.unisa.casper.analysis.code_smell_detection.similarityComputation	100% (1/1)	100% (9/9)	99% (949/952)	91% (33/36)
it.unisa.casper.analysis.code_smell_detection.misplaced_class	100% (2/2)	100% (6/6)	94% (79/84)	83% (20/24)
it.unisa.casper.refactor.splitting_algorithm	83% (5/6)	88% (24/27)	32% (445/1372)	73% (198/270)
it.unisa.casper.analysis.code_smell	72% (8/11)	54% (13/24)	60% (42/69)	57% (8/14)
it.unisa.casper.analysis.code_smell_detection.blob	66% (2/3)	66% (8/12)	55% (56/101)	55% (29/52)
it.unisa.casper.analysis.code_smell_detection.feature_envy	66% (2/3)	46% (6/13)	58% (91/156)	54% (26/48)
it.unisa.casper.structuralMetrics	100% (1/1)	48% (12/25)	58% (129/222)	50% (70/140)
it.unisa.casper.analysis.code_smell_detection	100% (2/2)	100% (9/9)	81% (36/44)	50% (27/54)
it.unisa.casper.storage.beans	100% (10/10)	69% (88/127)	66% (197/297)	24% (29/120)
it.unisa.casper.refactor.splitting_algorithm.irEngine	100% (4/4)	78% (18/23)	57% (414/719)	6% (26/416)
it.unisa.casper.gui	0% (0/63)	0% (0/198)	0% (0/2789)	0% (0/727)
it.unisa.casper.topic	0% (0/1)	0% (0/24)	0% (0/172)	0% (0/378)
it.unisa.casper.refactor.manipulator	0% (0/12)	0% (0/65)	0% (0/719)	0% (0/266)
it.unisa.casper.parser	0% (0/3)	0% (0/16)	0% (0/241)	0% (0/94)
it.unisa.casper.analysis.history_analysis_utility	0% (0/4)	0% (0/27)	0% (0/141)	0% (0/86)
it.unisa.casper.gui.charts	0% (0/4)	0% (0/13)	0% (0/133)	0% (0/38)
it.unisa.casper.statistics	66% (2/3)	7% (2/28)	1% (3/154)	0% (0/37)

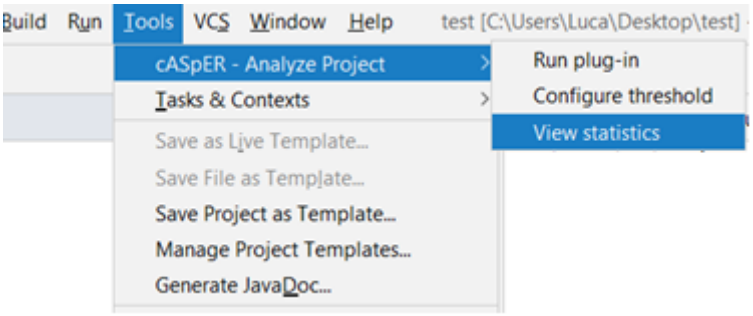
it.unisa.casper.analysis.code_smell_detection.shotgun_surgery	0% (0/1)	0% (0/6)	0% (0/64)	0% (0/28)
it.unisa.casper.analysis.code_smell_detection.parallel_inheritance	0% (0/1)	0% (0/7)	0% (0/64)	0% (0/26)
it.unisa.casper.analysis.code_smell_detection.divergent_change	0% (0/1)	0% (0/4)	0% (0/52)	0% (0/18)
it.unisa.casper.actions	0% (0/6)	0% (0/10)	0% (0/91)	0% (0/14)
it.unisa.casper.gui.radarMap	0% (0/2)	0% (0/10)	0% (0/48)	0% (0/4)

Il report mostra risultati eterogenei in quanto le classi responsabili delle funzionalità critiche del tool risultano sufficientemente coperte sia dal punto di vista dei metodi che dei branch, mentre quelle caratterizzate da un rischio più basso non sono coperte in buona parte. In realtà sappiamo che ottenere il 100% di copertura non sempre implica che il testing sia stato portato a termine in maniera efficace, in quanto riflette solo la quantità di codice eseguito. Possiamo dunque ritenerci soddisfatti dei risultati ottenuti e concludere il processo di testing.

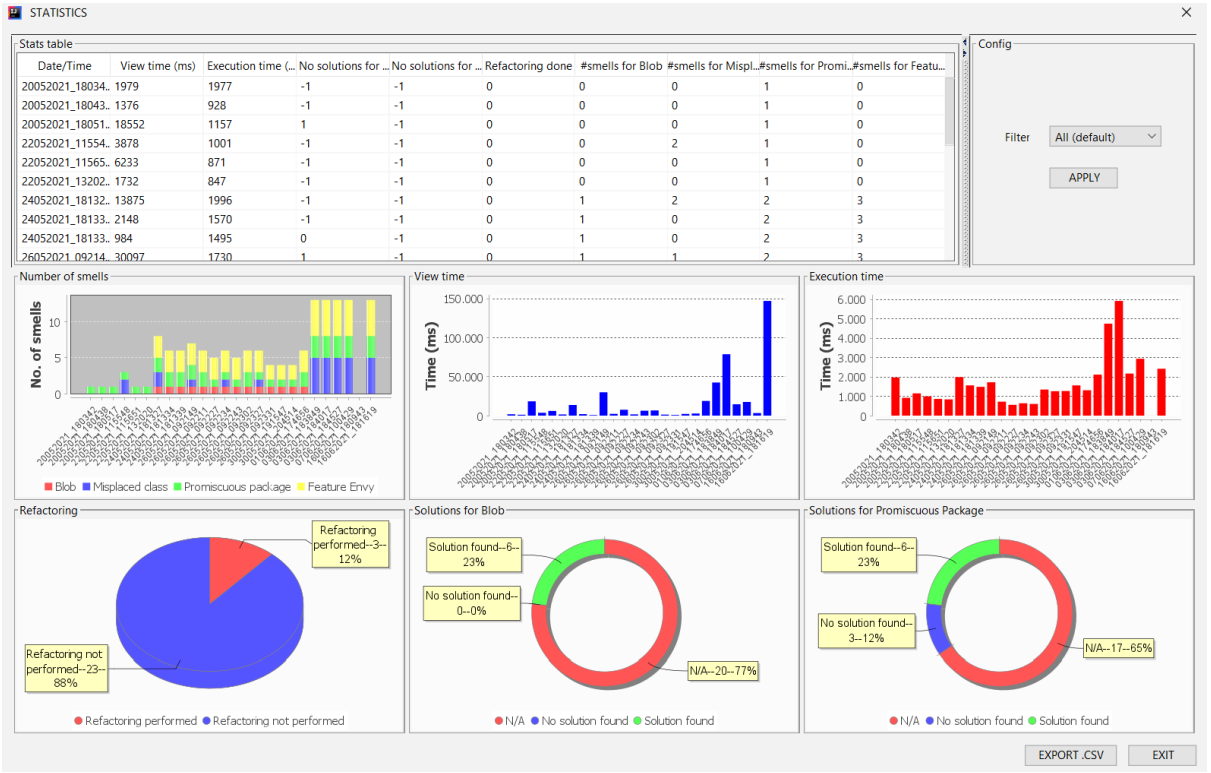
5 Conclusioni

Il sottosistema implementato raccoglie e permette di visualizzare alcune statistiche di utilizzo del tool.
Durante ogni esecuzione viene misurato il tempo che il tool impiega per effettuare l'analisi del codice (M1) e il tempo di utilizzo del tool da parte dell'utente, ovvero il tempo che intercorre tra la visualizzazione della prima interfaccia e la chiusura della finestra del tool (M4) Viene memorizzato il numero totale di smell che il tool riesce a riconoscere, distinguendoli per categoria (M2). Vengono conteggiate le volte in cui l'utente utilizza il tool per effettuare l'analisi del codice ma decide di non sfruttare la funzione di refactoring automatico (M3). Vengono conteggiate le volte in cui il tool non riesce a trovare una soluzione per gli smell che ha identificato (M5).

Per visualizzare le statistiche raccolte durante l'utilizzo del tool bisogna aprire il menù a tendina relativo a cASpER e selezionare la voce *cASpER - Analyze Project > View statistics*



Si aprirà una finestra contenente una tabella e alcuni grafici. La tabella mostra, in ogni riga, le statistiche raccolte durante ciascun utilizzo del tool. L'interfaccia permette di visualizzare in tabella un sottoinsieme delle esecuzioni totali. I grafici migliorano la leggibilità dei dati riportati in tabella permettendo all'utente di effettuare rapidi confronti e analisi, senza bisogno di esportare, tramite l'apposito pulsante, il file in formato ".csv". Esportare il file può essere utile per manipolare e analizzare i dati con l'ausilio di un software esterno.



Screenshot della pagina delle statistiche