**Università Degli Studi Di Salerno**

**Dipartimento di Informatica**

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

****

INGEGNERIA, GESTIONE ED EVOLUZIONE DEL

SOFTWARE

ANNO ACCADEMICO 2021/22

**Evoluzione cASpER**

Sommario

1. [Scopo](#Scopo)
2. [Panoramica del sistema](#PanoramicaSistema)
   1. [Requisiti Funzionali](#RequisitiFunzionali)
   2. [Design](#Design)
3. [Analisi delle modifiche richieste](#AnalisiModifiche)
4. [Primo intervento di manutenzione](#PrimoIntervento)
   1. [Analisi Change Request](#AnalisiCR1)
   2. [Impact Analysis](#ImpactAnalysisCR1)
      1. [Start Impact Set](#SISCR1)
      2. [Candidate Impact Set](#CISCR1)
   3. [Strategia di test](#StrategiTest)
   4. [Design](#DesignCR1)
   5. [Implementazione](#ImplementazioneCR1)
   6. [System Test](#SystemTestCR1)
5. [Secondo intervento di manutenzione](#SecondoInterventoManutenzione)
   1. [Analisi Change Request](#AnalisiCR2)
   2. [Impact Analysis](#ImpactCR2)
      1. [Start Impact Set](#SISCR2)
      2. [Candidate Impact Set](#CISCR2)
   3. [Strategia di test](#StrategiaTestCR2)
   4. [Design](#DesignCr2)
   5. [Implementazione](#ImplementazioenCR2)
   6. [System Test](#SystemTestCR2)
6. [Studio di fattibilità](#Fattibilita)
7. **Scopo**

In questo documento saranno descritti gli obiettivi del processo di estensione del plugin cASpER:

*Automated code Smell dEtection and Refactoring.*

Si studierà come le modifiche identificate impattano sugli artefatti del sistema esistente, per ognuna di esse è stato condotto uno studio di fattibilità apposito.

Infine, tale documento include la pianificazione del testing delle componenti che derivano dalle CR, con la descrizione di come viene effettuato il Regression Testing.

1. **Panoramica del sistema attuale**

cASpER (*Automated code Smell dEtection and Refactoring*) è un plugin Intellij che permette la rilevazione, attraverso le tecniche presenti nello stato dell’arte di code smell e permette di utilizzare le operazioni di refactoring provviste dalle API integrate di IntelliJ per risolvere i code smell rilevati.

Essendo che per il profetto cASpER non risulta presente alcuna documentazione a riguardo è stata applicata una fase di reverse engineering, in modo da riuscire a comprendere l’architettura ed i requisiti del sistema.

* 1. **Requisiti Funzionali**

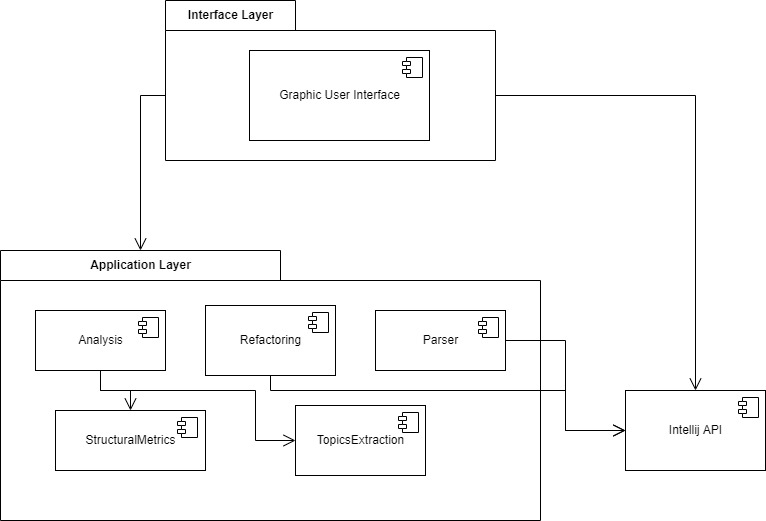
|  |  |  |
| --- | --- | --- |
| ID | Descrizione | Priorità |
| Requisiti per il rilevamento dei code smell | | |
| RF\_RCS\_1 | Il sistema dovrà supportare il rilevamento del code smell Blob | Alta |
| RF\_RCS\_2 | Il sistema dovrà supportare il rilevamento del code smell Feature Envy | Alta |
| RF\_RCS\_3 | Il sistema dovrà supportare il rilevamento del code smell MisplacedClass | Alta |
| RF\_RCS\_4 | Il sistema dovrà supportare il rilevamento del code smell Parallel Inheritance | Alta |
| RF\_RCS\_5 | Il sistema dovrà supportare il rilevamento del code smell Promiscuous Package | Alta |
| Rf\_RCS\_6 | Il sistema dovrà supportare il rilevamento del code smell Shotgun Surgery | Alta |
| RF\_RCS\_7 | Il sistema dovrà supportare il rilevamento del code smell Divergent Change |  |
| Requisiti per il refactoring dei code smell | | |
| RF\_RFCS\_1 | Il sistema dovrà supportare il refactoring del code smell Blob | Media |
| RF\_RFCS\_2 | Il sistema dovrà supportare il refactoring del code smell Feature Envy | Media |
| RF\_RFCS\_3 | Il sistema dovrà supportare il refactoring del code smell Misplaced Class | Media |
| RF\_RFCS\_4 | Il sistema dovrà supportare il refactoring del code smell Parallel Inheritance | Media |
| RF\_RFCS\_5 | Il sistema dovrà supportare il refactoring del code smell Shotgun Surgery | Media |
| RF\_RFCS\_6 | Il sistema dovrà supportare il refactoring del code smell Divergent Change | Media |
| RF\_RFCS\_7 | Il sistema dovrà supportare il refactoring del code smell Promiscuous Package | Media |
| Requisiti per la visualizzazione dei code smell rilevati e del loro eventuale refactoring | | |
| RF\_VLCSR\_1 | L’utente deve avere la possibilità di visualizzare i code smell presenti nel project analizzato | Alta |
| RF\_VCSR\_1 | L’utente deve avere la possibilità di visualizzare in modo dettagliato il code smell rilevato | Media |
| RF\_VCSRF\_1 | L’utente deve avere la possibilità di poter visualizzare l’anteprima del refactoring | Media |

* 1. **Design**

Il sistema è stato progettato in maniera tale da essere modulare; infatti esso possiede due layer differenti:

* Interface Layer: Questo livello contiene tutta la logica necessaria per mostrare le informazioni all’utente e dare la possibilità di interagire con il sistema
* Application Layer: Questo livello contiene tutta la logica del plugin, ovvero la possibilita di identificare gli smell ed effettuare il refactoring.

Il sistema software è stato sviluppato utilizzando il linguaggio di programmazione Object Oriented Java.



Come si può notare dal diagramma, le componenti destinate alla logica di presentazione, e quindi all’Interface Layer e le componenti destinati alla logica core, e quindi all’Application Layer sono due componenti distinte e separete. I due layer si compongono delle seguenti componenti:

* + 1. Interface Layer
* Graphic User Interface: Contiene la classe CheckProjectPaje che mostra all’utente gli smell presenti nel progetto analizzato. Inoltre contiene le classi per la generazione delle interfacce Page e Wizard, ovvero le interfacce per visualizzare in dettaglio lo smell rilevato e per visualizzare l’anteprima del refactoring
  + 1. Application Layer
* Analysis: Contiene la logica relativa alle regole per identificare i code smell. Ovviamente le rule fanno riferimento ai tipi di smell che il pulg-in va a considerare
* Refactoring: Contiene la logica relativa al refactoring dei code smell identificati
* Parser: Contiene la logica per convertire le componenti Psi, e esegue il processo di analisi sulle componenti del progetto.

1. **Analisi delle modifiche richieste**

|  |  |
| --- | --- |
| **Change\_Request\_id** | CR\_00\_Reengineering |
| **Priorità** | High |
| **Descrizione** | Reengineering della struttura di cASpER |
| **Tipo di Manutenzione** | Perfective |
| **Scopo** | Lo scopo della modifica e semplificare l’inserimento di nuovi Code Smell applicando dei design pattern |
| **Prodotto** | cASpER |

|  |  |
| --- | --- |
| **Change\_Request\_id** | CR\_01\_SpaghettiCodeDetection |
| **Priorità** | Normal |
| **Descrizione** | Implementazione del modulo di detection per il code smell “Spaghetti Code” |
| **Tipo di manutenzione** | Perfective |
| **Scopo** | Lo scopo della modifica è aggiungere al sistema un nuovo requisito che permette di identificare il code smell Spaghetti Code. Inoltre dovrà essere sviluppata anche la GUI relativa al code smell introdotto |
| **Prodotto** | cASpER |

|  |  |
| --- | --- |
| **Change\_Request\_id** | CR\_02\_SwissArmyKnifeDetection |
| **Priorità** | Normal |
| **Descrizione** | Implementazione del modulo di detection per il code smell “Swiss Army Knife” |
| **Tipo di Manutenzione** | Perfective |
| **Scopo** | Lo scopo della modifica è aggiungere al sistema un nuovo requisito che permette di identificare il code smell Spaghetti Code. Inoltre dovrà essere sviluppata anche la GUI relativa al code smell introdotto |
| **Prodotto** | cASpER |

L’intervento di manutenzione sarà diviso in due parti. Sarà prima eseguita la CR\_00\_Reengineering, successivamente poi saranno eseguite contemporaneamente le CR\_01\_SpaghettiCode e CR\_02\_SwissArmyKnifeDetection in quanto l’impact analysis, il processo di manutenzione e la fase di testing risulta essere la stessa.

1. **Primo intervento di manutenzione**

|  |  |
| --- | --- |
| **Change\_Request\_id** | CR\_00\_Reengineering |
| **Priorità** | High |
| **Descrizione** | Reengineering della struttura di cASpER |
| **Tipo di Manutenzione** | Perfective |
| **Scopo** | Lo scopo della modifica e semplificare l’inserimento di nuovi Code Smell applicando dei design pattern |
| **Prodotto** | cASpER |

**4.1** **Analisi Change Request**

Prima di eseguire l’intervento di manutenzione è stata eseguita un analisi della change request con lo scopo di identificare la fattibilità della richiesta.

Nella change request viene richiesto di eseguire il reengineering del sistema con lo scopo di semplificare l’inserimento di nuovi Code Smell da rilevare applicando dei design pattern. Pertanto abbiamo eseguito un analisi del sistema per poter capire come avviene l’inserimento di un code smell da rilevare e se essa potrebbe essere semplificata.

L’inserimento di un nuovo code smell comprende i seguenti passi:

1. Creare la classe relativa al code smell in cui salvare le varie info
2. Creare la classe in cui viene implementate la strategy per la detection del nuovo code smell
3. Inserire la detection del nuovo code smell nell modulo di parser
4. Creare la classe per la strategy del refactoring del nuovo code smell
5. Creare la GUI Wizard e Page del nuovo code smell
6. Aggiungere il nuovo code smell alla GUI CheckProjectPage

La procedura d’inserimento di un nuovo code smell risulta essere abbastanza semplice, tuttavia abbiamo notato che per ogni code smell viene eseguita l’operazione di detection. Tale procedura in base alla strategy specificata verifica se il bean e affetto da tale smell.

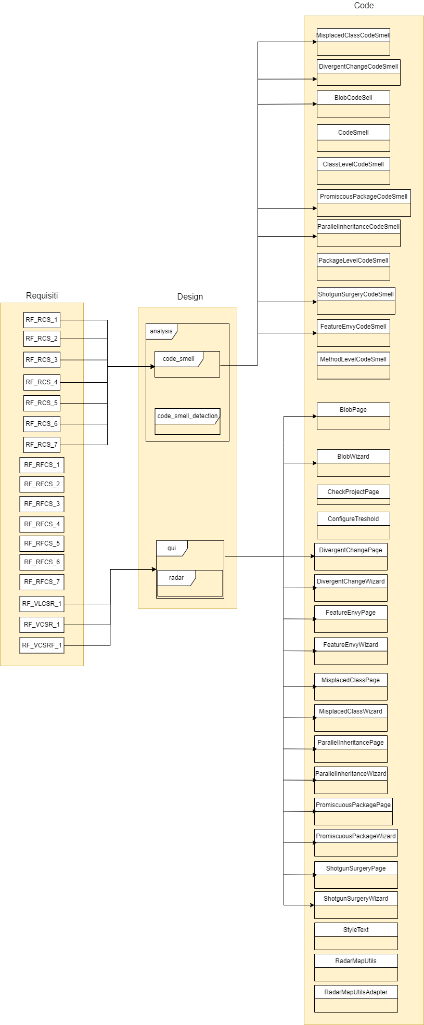
Abbiamo deciso di utilizzare il design pattern Visitor, in modo da separare l’operazione dal code smell.

Per quanto riguarda la GUI, ogni code smell possiede la propria classe Page e Wizard, tuttavia la classe CheckProjectPage risulta fortemente accoppiata a tali classi, abbiamo quindi deciso di applicare il design pattern “Abstract Factory” in modo da ridurre tale accoppiamento.

**4.2** **Impact Anlaysis**

**4.2.1** **Start Impact Set (SIS)**

L’identificazione del SIS è avvenuta grazie all’utilizzo del grafo tracciabilità:



Documentazione Generale/Immagini/TraceabilityGraph\_CR\_00\_Reegineering

Siamo partiti dai requisiti inserendo nell’omonima pila tutti i requisiti funzionali offerti da cASpER. Di tali requisiti nella change request vengono impattati solamente quelli relativi al rilevamento dei code smell, quelli relativi alla visualizzazione dei code smell presenti e alla visualizzazione dei refactoring.

Pertanto nella pila dedicata al design sono stai inseriti solamente i moduli che gestiscono tali requisiti, ovvero, Analysis e GUI. La tracciabilità orizzontale è stata mantenuta tra **RF\_VLCSR\_1**, **RF\_VCSR\_1, RF\_VCSRF\_1** e l’intero modulo GUI. Mentre per **RF\_RCS\_1**, **RF\_RCS\_2,** **RF\_RCS\_3**, **RF\_RCS\_4**, **RF\_RCS\_5**, **Rf\_RCS\_6**, **RF\_RCS\_7** la tracciabilità orizzontale è stata mantenuta solamente col sotto modulo code smell del modulo analysis in quanto il pattern visitor vuole essere applicato all’operazione di detection che viene eseguita nel modulo code smell, mentre il modulo code smell detection gestisce solamente le strategie di detection.

Dopo aver ottenuto la tracciabilità orizzontale tra la pila dei requisiti e quella del design siamo passati alla pila del codice, inserendo in essa tutte le classi del modulo code smell e GUI. Per la tracciabilità orizzontale tra la pila di design e codice sono stati escluse le classi Code Smell, ClassLevelCodeSmell, MethodLevelCodeSmell, PackageLevelCodeSmell, CheckProjectPage, ConfigureTreshold, StyleText, RadarMapUtils e RadarMapUtilsAdapter. Le prime tre vengono escluse in quanto non implementano il metodo per la detect degli smell ma viene implementato nelle classi che le estendono, quindi applicando il pattern visitor non saranno impattate. Tutte le altre vengono escluse in quanto non saranno impattate dal pattern Abstract Factory poiché esso lavora solamente sulle classi Wizard e Page.

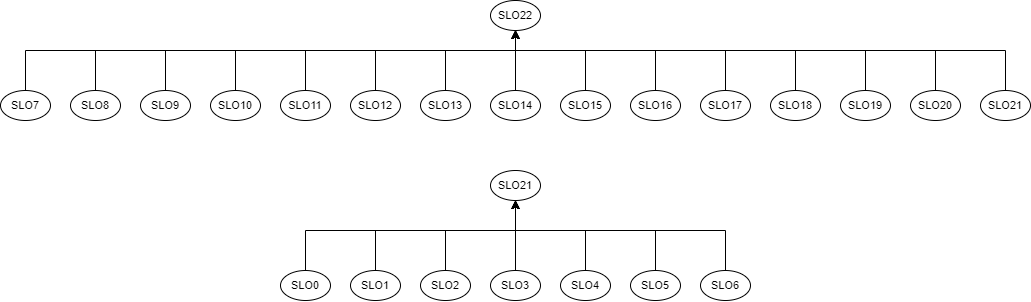
Dopo la pila del codice dovrebbe essere presente la pila dei test per capire quali test vengono impattati dalla CR, tuttavia in cASpER non è implementato nessun test per per la detection dei code smell ne per la GUI.

In conclusione lo start impact set (SIS) risulta essere il seguente, {**MisplacedClassCodeSmell, DivergentChangeCodeSmell, BlobCodeSmell, PromiscuousPackageCodeSmell, ParallelInheritanceCodeSmell, ShotgunSurgeryCodeSmell, FeatureEnvyCodeSmell, BlobPage, BlobWizard, DivergentChangePage, DivergentChangeWizard, FeatureEnvyPage, FeatureEnvyWizard, MisplacedClassPage, MisplacedClassWizard, ParallelInheritancePage, ParallelInheritancheWizard, PromiscuousPackagePage, PromiscuousPackageWizard, ShotgunSurgeryPage, ShotgunSurgeryWizard**}

**4.2.2** **Candidate Impact Set (CIS)**

Per l’identificazione del CIS siamo andati a rilevare analizzando il codice i possibili impatti diretti e indiretti delle classi presenti all’interno del SIS.

|  |  |
| --- | --- |
| * SLO0 - MisplacedClassCodeSmell * SLO1 - DivergentChangeCodeSmell * SLO2 - BlobCodeSmell * SLO3 - PromiscuousPackageCodeSmell * SLO4 - ParallelInheritanceCoseSmell * SLO5 - ShotgunSurgeryCodeSmell * SLO6 - FutureEnvyCodeSmell * SLO7 - BlobPage * SLO8 - BlobWizard * SLO9 - DivergentChagePage * SLO10 - DivergentChangeWizard * SLO11 - FutureEnvyPage | * SLO12 - FutureEnvyWizard * SLO13 - MisplacedClassPage * SLO14 - MisplacedClassWizard * SLO15 - ParallelInheritancePage * SLO16 - ParallelInheritanceWizard * SLO17 - PromiscuousPackagePage * SLO18 - PromiscuousPackageWizard * SLO19 - ShotgunSurgeryPage * SLO20 - ShotgunSugeryWizard * SLO21 - PsiParser * SLO22 – CheckProjectPage |

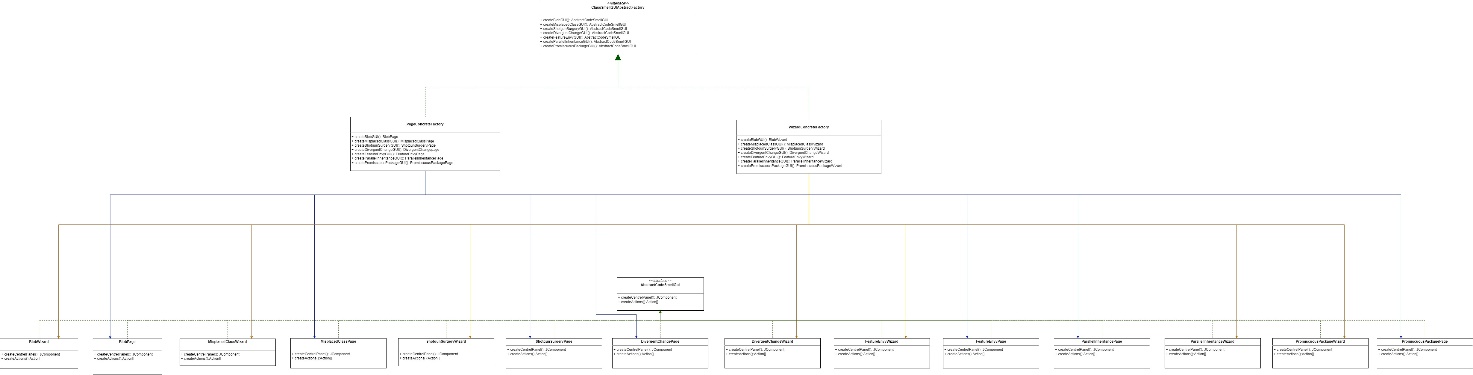


Documentazione Generale/Immagini/Grafo di Raggiungibilita\_CR\_00\_Reeginering

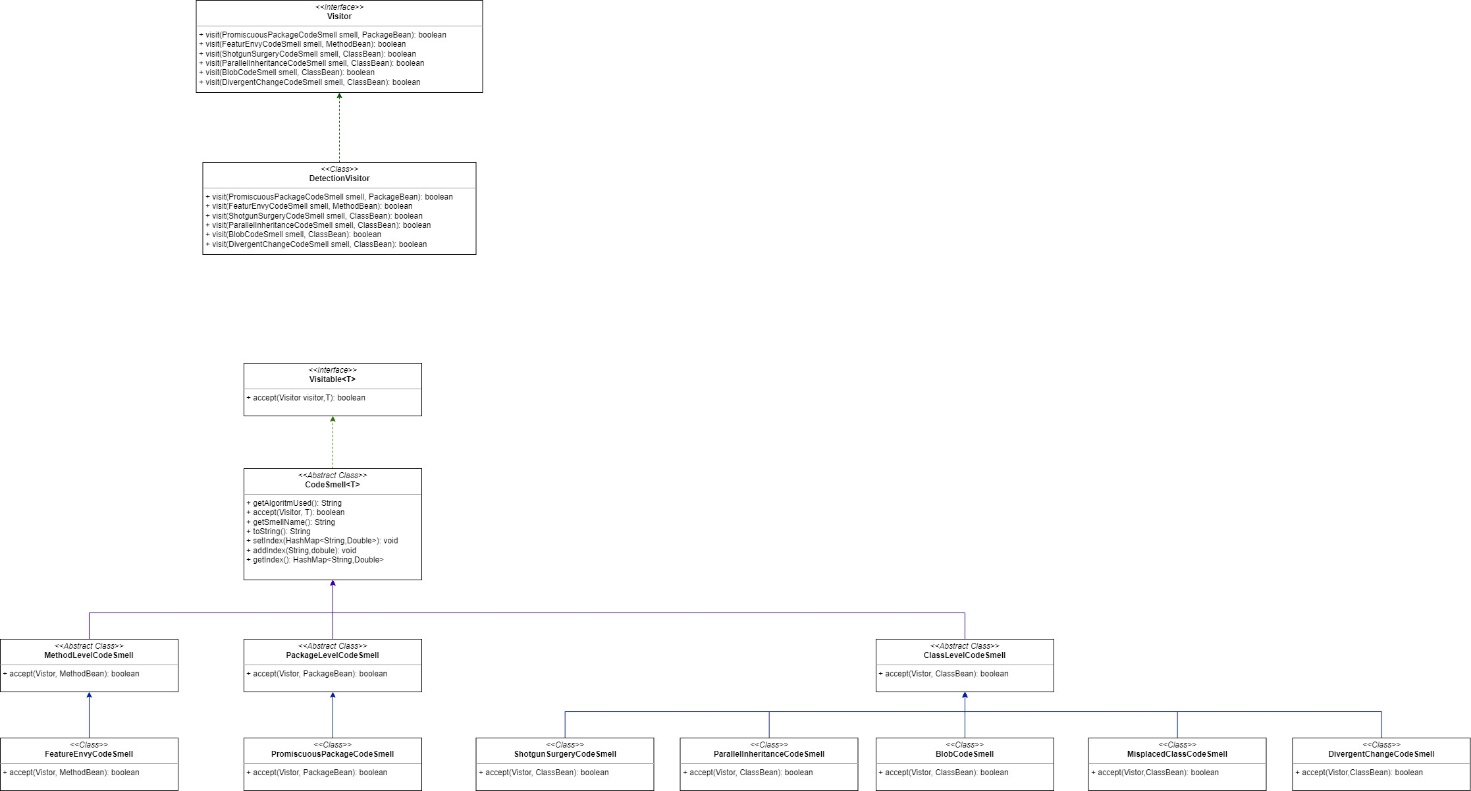
Il candidate impact set previsto per l implementazione di tale change request risulta essere il seguente, {**MisplacedClassCodeSmell, DivergentChangeCodeSmell, BlobCodeSmell, PromiscuousPackageCodeSmell, ParallelInheritanceCodeSmell, ShotgunSurgeryCodeSmell, FeatureEnvyCodeSmell, BlobPage, BlobWizard, DivergentChangePage, DivergentChangeWizard, FeatureEnvyPage, FeatureEnvyWizard, MisplacedClassPage, MisplacedClassWizard, ParallelInheritancePage, ParallelInheritancheWizard, PromiscuousPackagePage, PromiscuousPackageWizard, ShotgunSurgeryPage, ShotgunSurgeryWizard, PsiParser, CheckProjectPage**}

**4.3 Design**

In tale fase sono state implementate le modifiche a livello di design. Pertanto saranno applicati, il design pattern “Abstract Factroy” al modulo per la gestione delle GUI e il design pattern “Visitor” al modulo per la detection dei code smell.



Documentazione Generale/Immagini/AbstractFactory GUI



Documentazione Generale/Immagini/Visitor

Da come si può notare dalla seguente immagine, per l’implementazione del visitor, l’interfaccia Visitable viene implementata direttamente dalla classe CodeSmell. Tuttavia tale classe non è presente nel CIS, pertanto si può già prevedere che la recall della impact analysis non sarà uguale ad 1.

**4.5** **Implementazione**

Prima di eseguire l’implementazione della modifica richiesta è stato eseguito il test di regressione. Tuttavia tale test è stato rieseguito anche in seguito alla manutenzione, così da poter capire se l’intervento di manutenzione abbia introdotto nuove failure che non erano presenti prima di esso.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Classi | SIS | CIS | AIS | DIS | FPIS |
| ClassLevelCodeSmell |  |  | X | X |  |
| CodeSmell |  |  | X | X |  |
| MethodLevelCodeSmell |  |  | X | X |  |
| PackageLevelCodeSmell |  |  | X | X |  |
| DivergentChangeCodeSmell | X | X | X |  |  |
| BlobCodeSmell | X | X | X |  |  |
| PromiscuousPackageCodeSmell | X | X | X |  |  |
| ParallelInheritanceCodeSmell | X | X | X |  |  |
| ShotgunSurgeryCodeSmell | X | X | X |  |  |
| FutureEnvyCodeSmell | X | X | X |  |  |
| BlobPage | X | X | X |  |  |
| BlobWizard | X | X | X |  |  |
| DivergentChangePage | X | X | X |  |  |
| DivergentChangeWizard | X | X | X |  |  |
| FutureEnvyPage | X | X | X |  |  |
| FutureEnvyWizard | X | X | X |  |  |
| MisplacedClassPage | X | X | X |  |  |
| MisplacedClassWizard | X | X | X |  |  |
| ParallelInheritancePage | X | X | X |  |  |
| ParallelInheritanceWizar | X | X | X |  |  |
| PromiscuousPackagePage | X | X | X |  |  |
| PromiscuousPackageWizard | X | X | X |  |  |
| ShotgunSurgeryPage | X | X | X |  |  |
| ShotgunSurgeryWizard | X | X | X |  |  |
| PsiParser |  | X | X |  |  |
| CheckProjectPage |  | X | X |  |  |
| ClassBean |  |  | X | X |  |
| MethodBean |  |  | X | X |  |
| PackageBean |  |  | X | X |  |

**Recall Impact Analysis:** |CIS ∩ AIS| /|AIS| = 23/30 = 0,76

**Precision Impact Analysis:** |CIS ∩ AIS| /|CIS| = 23/23 = 1

1. **Secondo intervento di manutenzione**

|  |  |
| --- | --- |
| **Change\_Request\_id** | CR\_01\_SpaghettiCodeDetection |
| **Priorità** | Normal |
| **Descrizione** | Implementazione del modulo di detection per il code smell “Spaghetti Code” |
| **Tipo di manutenzione** | Perfective |
| **Scopo** | Lo scopo della modifica è aggiungere al sistema un nuovo requisito che permette di identificare il code smell Spaghetti Code. Inoltre dovrà essere sviluppata anche la GUI relativa al code smell introdotto |
| **Prodotto** | cASpER |

|  |  |
| --- | --- |
| **Change\_Request\_id** | CR\_02\_SwissArmyKnifeDetection |
| **Priorità** | Normal |
| **Descrizione** | Implementazione del modulo di detection per il code smell “Swiss Army Knife” |
| **Tipo di Manutenzione** | Perfective |
| **Scopo** | Lo scopo della modifica è aggiungere al sistema un nuovo requisito che permette di identificare il code smell Spaghetti Code. Inoltre dovrà essere sviluppata anche la GUI relativa al code smell introdotto |
| **Prodotto** | cASpER |

* 1. **Analisi Change Request**

Anche per il secondo intervento di manutenzione, prima di intraprendere tale processo, è stata eseguita un’analisi della change request, per valutarne la fattibilità.

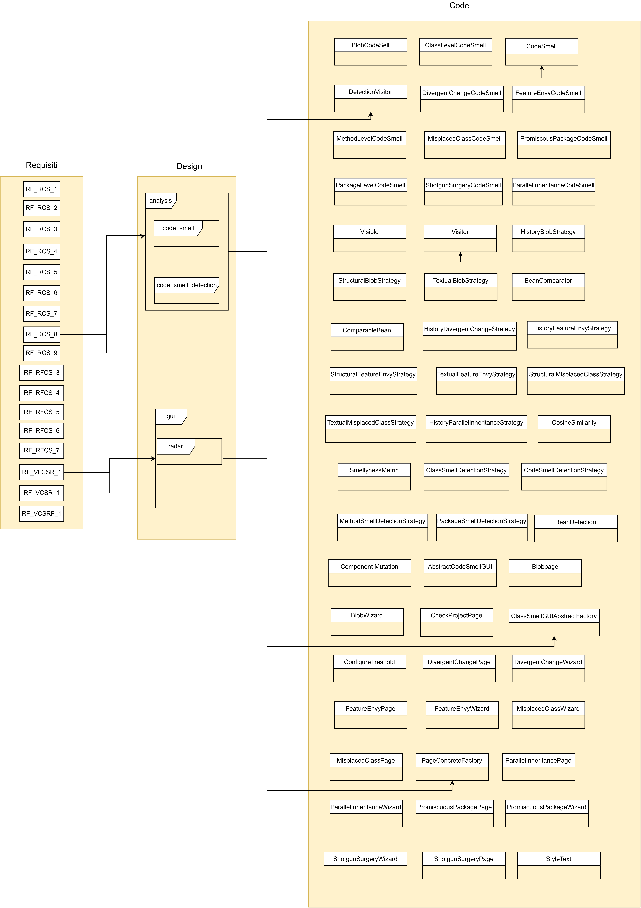
Le change request hanno lo scopo di aggiungere due nuovi requisiti al sistema:

|  |  |  |
| --- | --- | --- |
| ID | Descrizione | Priorità |
| Requisiti per il rilevamento dei code smell | | |
| RF\_RCS\_8 | Il sistema dovrà supportare il rilevamento del code smell Spaghetti Code | Alta |
| RF\_RCS\_8 | Il sistema dovrà supportare il rilevamento del code smell SwissArmyKnife | Alta |

Inoltre dovrà essere modificata la GUI in modo da introdurre tale code smell, ma essendo che la change request tratta solamente la detection, non sarà implementata la GUI che mostra l’anteprima del refactoring.

* 1. **Impact Anlaysis**
     1. **Start Impact Set (SIS)**

L’impact analysis è iniziata col costruire un grafo della tracciabilità andando ad analizzare gli artefatti disponibili di cASpER. Il grafo risultante e il seguente:



Documentazione Generale/Immagini/TraceabilityGraph\_CR\_01-CR\_02

Siamo partiti dai requisiti inserendo nell’omonima pila tutti i requisiti funzionali offerti da cASpER compresi i nuovi requisiti da implementare. Dai nuovi requisiti da implementare inizia la tracciabilità orizzontale. Pertanto nella pila dedicata al design sono stai inseriti solamente i moduli che gestiscono tali requisiti, ovvero, Analysis e GUI. La tracciabilità orizzontale è stata mantenuta tra **RF\_VLCSR\_1**,**RF\_VCSR\_1** e l’intero modulo GUI. Mentre per **RF\_RCS\_8** e **RF\_RCS\_9** la tracciabilità orizzontale è stata mantenuta col modulo analysis. In quanto tale modulo gestisce la detection dei code smell.

Dopo aver ottenuto la tracciabilità orizzontale tra la pila dei requisiti e quella del design siamo passati alla pila del codice, inserendo in essa tutte le classi del modulo analysis e GUI. Le uniche classi che vengono impattate dall’aggiunta di un nuovo code smell risultano essere CodeSmell, DetectionVisitor, Visitor, ClassSmellGUIAbstractFactory e PageConcreteFactory.

Dopo la pila del codice dovrebbe essere presente la pila dei test per capire quali test vengono impattati dalla CR, tuttavia in cASpER non è implementato nessun test relativo alle classi impattate, pertanto la pila dei test risulta essere vuota.

In conclusione lo start impact set (SIS) risulta essere il seguente, {**CodeSmell, DetectionVisitor, Visitor, ClassSmellGUIAbstractFactory, PageConcreteFactory**}

* + 1. **Candidate Impact Set (CIS)**

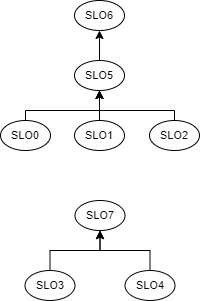
In tale fase integriamo il SIS con gli SLO, che possono cambiare a causa di modifiche agli elementi del SIS, analizzando gli impatti diretti e indiretti

|  |  |
| --- | --- |
| * SLO0 - CodeSmell * SLO1 - DetectionVisitor * SLO2 - Visitor * SLO3 - ClassSmellGUIAbstractFactory * SLO4 - PageConcreteFactory * SLO5 - PsiParser * SLO6 - ClassBean * SLO7 - CheckProjectPage |  |

Aggiungendo un nuovo CodeSmell andremo ad impattare direttamente sulla classe PsiParser, in quanto essa avvia la detection degli smell. Pertanto dovrà essere inserita anche la detection dei nuovi smell da implementare. Tuttavia per la rilevazione di SpaghettiCode e SwissArmyKnife, bisogna controllare se implementano più interfacce o estendono qualche classe. Attualmente la classe ClassBean costruita sulla classe PsiClass non mantiene traccia di tali informazioni.

Andremo a modificare la classe PsiParser in modo che quando si esegue il parse da PsiClass a ClassBean vengono salvate le classi implementate e le classi estese. Tale modifica quindi impatterà in maniera diretta sulla classe ClassBean poichè dovrà essere aggiunto un attributo per conservare le classi implementate ed uno per conservare le classi estese.

Inoltre anche la classe CheckProjectPage sarà impattata direttamente per l’aggiunta di una nuova page.



Documentazione Generale/Immagini/Grafo di Raggiungibilita\_CR\_01-CR\_02

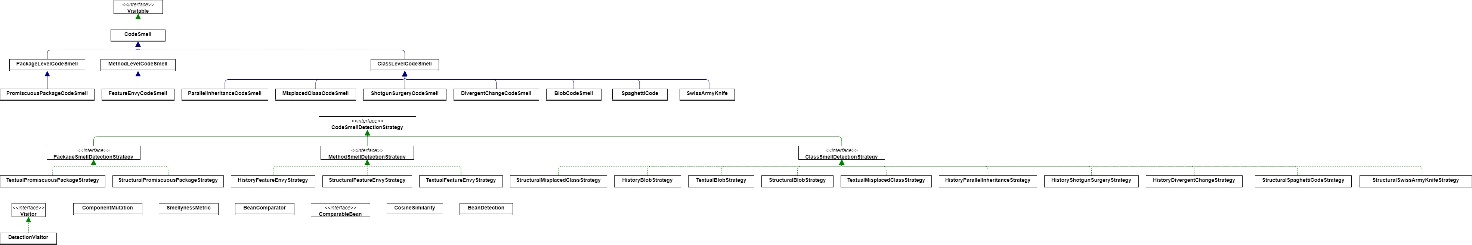
In conclusione il candidate impact set previsto per l’implementazione di tale change request risulta essere il seguente, {**CodeSmell, DetectionVisitor, Visitor, ClassSmellGUIAbstractFactory, PageConcreteFactory, PsiParser, ClassBean, CheckProjectPage** }

* 1. **Strategia di Test**

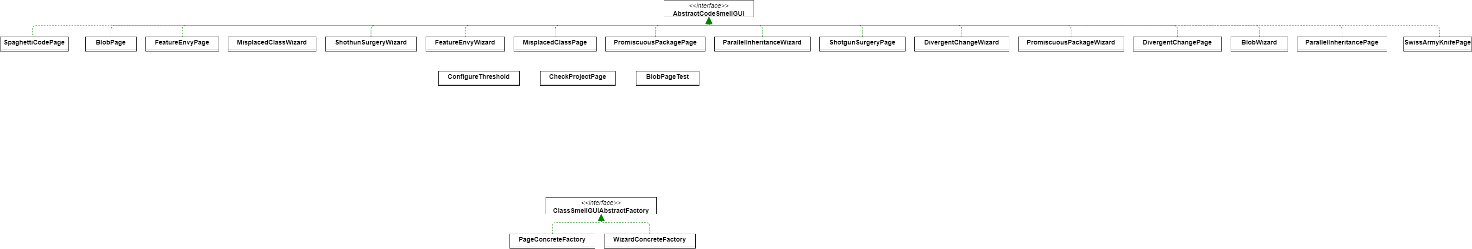
Dopo aver eseguito l’impact anlaysis abbiamo stabilito una strategia di test da applicare per l’intervento di manutenzione. Tutta la documentazione relativa al testing è presente nella cartella Testing\_CR\_01-CR\_02

* 1. **Design**

In tale fase sono state implementate le modifiche a livello di design.



Documentazione Generale/Immagini/Analysis



Documentazione Generale/Immagini/GUI

Successivamente all’implementazione delle modifiche a livello di design è stato scritto il Test Plan ed è stato identificato una Test Case Suite. Tutta la documentazione relativa al testing è presente nella cartella Testing\_CR\_01-CR\_02

* 1. **Implementazione**

Prima di eseguire l’implementazione della modifica è stato eseguito il test di regressione.

Per la detection degli smell Spaghetti Code e SwissArmyKnift bisogna conoscere le interfacce implementate e le classi estese da una classe, tuttavia il parser che converte un oggetto PsiClass in ClasseBean non tiene traccia di tali attributi, abbiamo deciso di modificare prima il parser è poi sviluppare la detection degli smells.

Dopo aver modificato il parser è stato eseguito il test d’unita su di esso, in modo da verificare il corretto funzionamento.

Per l’implementazione della detection degli smell abbiamo sfruttato le seguenti rule card:

|  |  |  |
| --- | --- | --- |
| Rule\_Card: | SpaghettiCode | |
| Rule: | SpaghettiCode | INTER LongMethod, NoParameter, NoInheritance, ProceduralName |
| Rule: | LongMethod | METRIC LongMethod 25 |
| Rule: | NoParameter | METRIC NMNOPARAM 0 |
| Rule: | NoInheritance | STRUCTURAL DIT 0 |
| Rule: | ProceduralName | LEXIC Class\_Name (Make, Create, Exectue…) |

|  |  |  |
| --- | --- | --- |
| Rule\_Card: | SwissArmyKnife | |
| Rule: | MultipleInterface | METRIC NINTERF 3 |

Dopo aver implementato le varie strategy e i detector per eseguire il rilevamento è stato eseguito il test di unita. Successivamente ad esso è stata testata l’integrazione dei detector con il parser che è responsabile di avviare l’analisi delle classi.

Per la modifica delle GUI è stata implementata solo la classe Page per gli smell introdotti mantenendo lo stesso layout di tutte le altre Page. Infine è stata modificata la classe CheckProjectPage in modo da integrare nella lista degli smell individuati anche i nuovi smell introdotti.

L’ultimo steps di tale fase è l’esecuzione dei test di regressione, in modo da poter capire se l’intervento di manutenzione abbia introdotto nuove failure che non erano presenti prima di esso.

* 1. **System Test**

In tale fase è stato eseguito il test di sistema ed i relativi risultati sono presenti nella cartella Testing\_CR\_01-CR\_02.

Oltre ad eseguire i test di sistemi, è stata valutata anche la recall e la precision dell’impact analysis.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Classi | SIS | CIS | AIS | DIS | FPIS |
| CodeSmell | X |  | X |  |  |
| DetectionVisitor | X |  | X |  |  |
| Visitor | X |  | X |  |  |
| ClassSmellGUIAbstractFactroy | X |  | X |  |  |
| PageConcreteFactory | X | X | X |  |  |
| PsiParser |  | X | X |  |  |
| ClassBean |  | X | X |  |  |
| CheckProjectPage |  | X | X |  |  |
| WizardConcreteFactory |  |  | X | X |  |
| ConfigureThreshold |  |  | X | X |  |
| SystemStart |  |  | X | X |  |

**Recall Impact Analysis:** |CIS ∩ AIS| /|AIS| = 8/11 = 0,72

**Precision Impact Analysis:** |CIS ∩ AIS| /|CIS| = 8/8 = 1

1. **Studio di Fattibilità**

Scala ordine di valutazione:

* Alta: Si intende che la maggior parte dello sforzo verrà impegato nel focus dell’attività
* Media: Si intende che buona parte dello sforzo verrà impegato nel focus dell’attività
* Bassa: Si intende lo sforzo minimo necessario allo svolgimento dell’attività
  1. Identificazione, Descrizione e Valutazione dei Costi

|  |  |  |
| --- | --- | --- |
| Identificazione | Valutazione | Descrizione |
| Sviluppo di novi code smell | Alta | La modifica richiesta prevede la creazione della logia di identificazione dei nuovi code smell |
| Organizzazione del lavoro nel team di sviluppo | Media | I componenti del team di sviluppo condividono poche ore settimanali per le modifiche da apportare al sistema. Pertanto, verranno effettuare riunioni anche nel fine settimana. |
| Progettazione testing | Alta | Il testing dei nuovi detector è di fondamentale importanza per capire se si è effettivamente in grado di rilevare lo smell in classi di progetti d’uso reale. |

* 1. Identificazione, Descrizione e Valutazione dei Benefici

|  |  |  |  |
| --- | --- | --- | --- |
| Identificazione | Valutazione | Descrizione | Strategia |
| Errori di identificazione dei code smell | Alta | Le operazioni di identificazione dei nuovi code smell potrebbero segnalare falsi positivi | Contingenza: Spendere più effort da parte dell’intero team nel testing per capire il malfunzionamento della detection e relativo bug fixing. |
| Moduli del sistema non funzionanti | Media | Alcuni moduli di supporto al progetto potrebbero non funzionare correttamente | Minimization:  se il modulo non è oggetto delle change requests in questione sarà ignorato. |
| Complessità del sistema da realizzare | Media | Il numero di funzionalità del software potrebbe aumentare. | Minimization:  eventuali funzionalità emerse durante la fase di sviluppo, se non di vitale importanza, saranno implementate nelle release future. |