Conduite de projet
Rapport du projet

Clément Badiola Samuel Da Silva Reda Lyazidi
Ladislav Marsik Alexandre Perrot
Client : Hugo Balacey

4 novembre 2012

Introduction

Ce document est le rapport d'un projet de conduite de projet effectué par un groupe de 5 étudiants dans le cadre de leur deuxième année de Master.

Le projet concerne blablabla. Il s'agissait de développer des fonctionnalités dans le cadre blablabla et des bonnes pratiques blabla.

Mots-clés : conduite de projet, agile, scrum, gestion

Table des matières

Introduction **i**

1 Gestion de projet **1**
1.1 Gestion de projet . . . 1
1.1.1 Pré-requis du projet . . . 1
1.1.2 Mise en place des outils . . . 1
1.1.3 Description du déroulement projet . . . 1

2 Cahier des charges **2**
2.1 Besoins non-fonctionnels . . . 2
2.1.1 exemple (refactoring . . . 2

3 Architecture **3**
3.1 Diagramme des cas d'utilisation . . . 3
3.2 Les classes . . . 5
3.2.1 Le modèle d'examen . . . 6
3.2.2 L'interface graphique . . . 6

4 Travail **8**
4.1 Travail . . . 8
4.1.1 L'explorateur de fichiers . . . 8
4.1.2 L'interaction avec un examen . . . 8
4.2 Tests . . . 8
4.2.1 Test 1 . . . 8

5 Bilan **9**
5.1 Bilan . . . 9
5.1.1 Échecs . . . 9
5.2 Si c'était à refaire. 9

Chapitre 1

Gestion de projet

Sommaire

1.1 Gestion de projet 1
1.1.1 Pré-requis du projet 1
1.1.2 Mise en place des outils 1
1.1.3 Description du déroulement projet 1

1.1 Gestion de projet

1.1.1 Pré-requis du projet

1.1.2 Mise en place des outils

1.1.3 Description du déroulement projet

Chapitre 2

Cahier des charges

Sommaire

2.1 Besoins non-fonctionnels 2
2.1.1 exemple (refactoring 2

2.1 Besoins non-fonctionnels

2.1.1 exemple (refactoring

Le refactoring consiste à retravailler un code source dans le but d'améliorer sa lisibilité et son efficacité, et de simplifier sa maintenance. L'introduction de nouveaux patterns induit le besoin de refactoriser souvent le code afin de simplifier le codage et la compréhension. En plus d'un simple nettoyage, cela nous amène à vérifier que notre architecture répond toujours aux objectifs fixés.

L'objectif est bien sûr d'obtenir un gain de clarté, de lisibilité, de maintenabilité, et probablement de performances. Nous pouvons ainsi continuer l'ajout de fonctions sur une base saine.

Chapitre 3

Architecture

Sommaire

3.1 Diagramme des cas d'utilisation 3
3.2 Les classes 5
3.2.1 Le modèle d'examen 6
3.2.2 L'interface graphique 6

Le domaine de l'imagerie médicale regorge de concepts nouveaux et d'algorithmes complexes. Afin de compenser cette difficulté, nous nous sommes attachés à plusieurs lignes directrices lors de la conception et la réalisation de notre projet, pour que l'application finale soit la plus simple possible.

- Garder une architecture la plus simple possible. Chaque classe représente quelque chose de précis.
- Ranger les méthodes dans les bonnes classes.
- Limiter les attributs des classes afin de limiter les bogues.
- Limiter les dépendances externes et au langage. Les langages évoluent vite et il peut être parfois intéressant de pouvoir passer une application sur un autre langage.

3.1 Diagramme des cas d'utilisation

Nous avons réalisé un schéma des cas d'utilisations, visible en figure 3.1 page suivante afin de synthétiser les besoins de l'utilisateur, pour offrir une vision simplifiée du système. Nous rappellerons brièvement les fonctionnalités par la suite.

Ouvrir un examen

L'utilisateur peut ouvrir un examen stocké sur sa tablette en parcourant l'arborescence de fichiers et en cliquant sur l'examen choisi.

Visualiser les informations d'un examen

Le client peut, n'importe où dans un examen, visualiser les informations associées à cet examen en cliquant sur un bouton qui déclenchera l'ouverture de

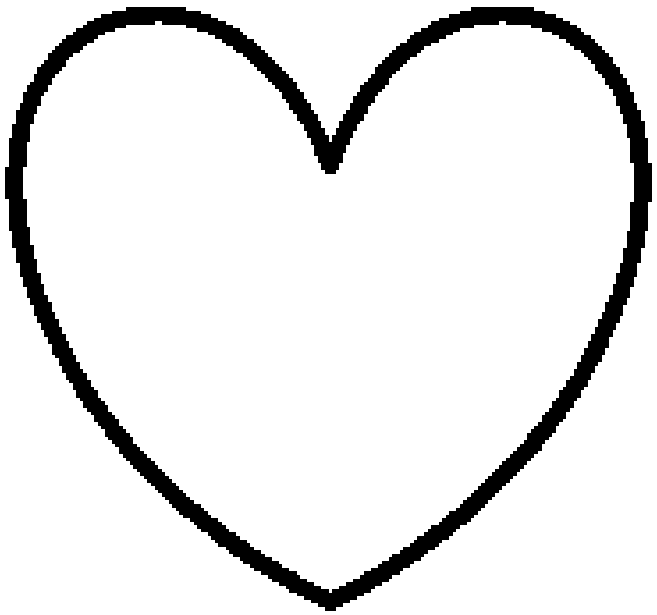

FIGURE 3.1 – Diagramme de cas d'utilisation

la fenêtre d'affichage des informations. Il pourra y voir des informations sur le patient et sur les conditions d'examen.

Visualiser les informations d'une coupe

Le client peut visualiser les informations associées à une coupe en cliquant sur un bouton qui déclenchera l'ouverture de la fenêtre d'affichage des informations. Il pourra y voir des informations associées à la coupe affichée à l'écran.

Naviguer entre les coupes

Le client peut visualiser les différentes coupes qui composent un examen qu'il a précédemment ouvert en cliquant sur les boutons associés aux déplacements entre coupes.

Dessiner des zones sur un masque de dessin

Le client peut dessiner sur un masque de dessin en superposition avec l'image d'une coupe de l'examen. Il dispose d'outils de dessin tels que le crayon ou la gomme.

Modifier les options de dessin

Le client peut dessiner sur un masque de dessin en variant les effets tels que l'épaisseur du trait utilisé.

Modifier le contraste de l'examen

Le client peut à tout moment changer le contraste (échelle de Hounsfield) des coupes de l'examen ouvert en utilisant les contrôles associés à cet effet. Il peut modifier la largeur de l'échelle et son centre. Il peut également choisir un contraste prédéfini parmi une liste de contrastes pour visualiser des éléments spécifiques tels que les os ou les chairs.

Se déplacer dans l'image

Le client peut visualiser différentes portions d'une image en la faisant glisser sur l'écran avec les doigts.

Changer le niveau de zoom d'une image

Le client peut modifier le niveau de zoom d'une image en utilisant le contrôle prévu à cet effet. Il peut augmenter ou diminuer le niveau de zoom.

Sauvegarder les modifications

Le client peut enregistrer ses modifications (dessin, contraste par défaut) effectuées sur un examen.

Restaurer les modifications

Le client peut charger ses modifications (dessin, contraste par défaut) effectuées auparavant sur un examen.

Appliquer un filtre

Le client peut modifier les images d'un examen en appliquant un filtre (tel que le flou gaussien).

Sélectionner des zones

Le client peut sélectionner des zones sur une coupe pour appliquer des traitements spécifiquement sur ces zones.

3.2 Les classes

Les diagrammes de classes de l'architecture du logiciel ont été réalisés à l'aide des logiciel *Architexa* et *UML Lab* intégrés à *Eclipse*. Nous ne parlerons donc que de notre architecture finale, obtenue au fur et à mesure des sprints et refactorings.

Étant donné l'étendue de l'application, nous ne pouvons pas présenter de diagramme UML à la fois complet et lisible dans ce rapport. Commençons donc par obtenir une vue d'ensemble du logiciel à l'aide du diagramme en couches

visible en figure 3.2. En bleu, les packages de transformation des données. En vert, les packages utilitaires. En rouge, le cœur de l'application, qui inclue dans des sous-packages le modèle des données, la gestion de cache et les classes d'interface graphique. Les flèches, plus ou moins épaisses, indiquent une dépendance plus ou moins forte d'un package à un autre. On remarque bien que le package diams constitue le cœur de l'application. Les packages utilitaires peuvent être changés sans problèmes car ils ne dépendant pas d'autres packages. Le package pixelmed constitue une bibliothèque de traitement de fichiers Dicom dont le cœur d'application dépend. L'enjeu majeur a donc été de rendre l'architecture du cœur d'application la plus flexible possible pour limiter l'effort requis en cas de changement de bibliothèques.

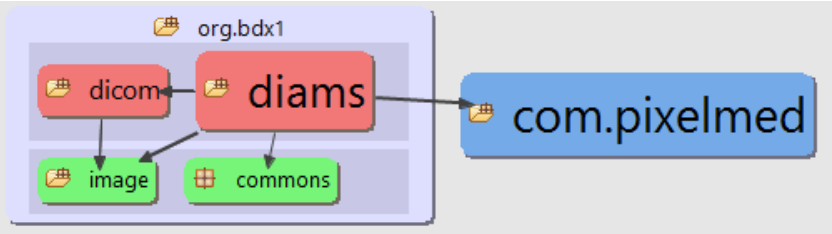

FIGURE 3.2 – Diagramme en couches de l'application

3.2.1 Le modèle d'examen

Voyons maintenant le diagramme des classes du modèle, en figure 3.3 page suivante. En bleu, les classes utilitaires, en vert le modèle qui représente un examen et les données associées, et en orange les éléments qui permettent de faire le lien entre le modèle et la bibliothèque d'extraction des images *Pixelmed*.

La classe `DefaultModelFactory` permet d'interagir depuis l'extérieur avec le modèle d'un examen sans dépendre des autres classes. La classe `Examen` est le composant principal qui permet de gérer les autres éléments du modèle. Un ensemble d'interfaces permet de faciliter des modifications ultérieures sur le modèle.

Notez que la classe orange `LisaImageAdapter` est le seul élément de dépendance avec la bibliothèque externe *Pixelmed*. Nous avons donc minimisé l'impact que peut créer le passage de *Pixelmed* à un autre outil.

3.2.2 L'interface graphique

Nous nous intéressons maintenant au diagramme des classes simplifié de la partie graphique, visible en figure 3.4 page suivante. En bleu, les classes d'interface graphique. En orange, les classes d'interaction avec les fichiers et en vert la classe d'interaction avec le modèle.

La classe `FileBrowserActivity` permet, à l'ouverture de l'application, de sélectionner et d'ouvrir un examen parmi une arborescence de fichiers. Les classes `ImageActivity` et `InfoDisplayActivity` permettent de visualiser les composants de l'examen choisi et d'interagir avec. La classe `DicomDirFileHandler`

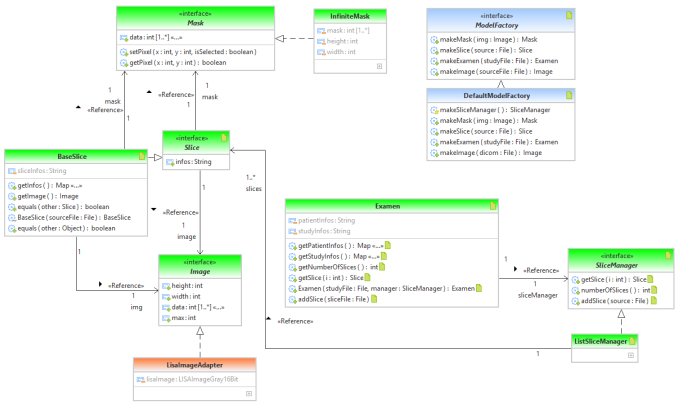

FIGURE 3.3 – Diagramme de classes du modèle de l'application

gère les traitements des fichiers lors de la navigation dans l'arborescence des fichiers.

On note que les dépendances entre l'interface graphique et les données sont limitées à deux classes. Nous avons minimisé l'impact d'un changement de bibliothèque d'interface graphique à ces deux classes.

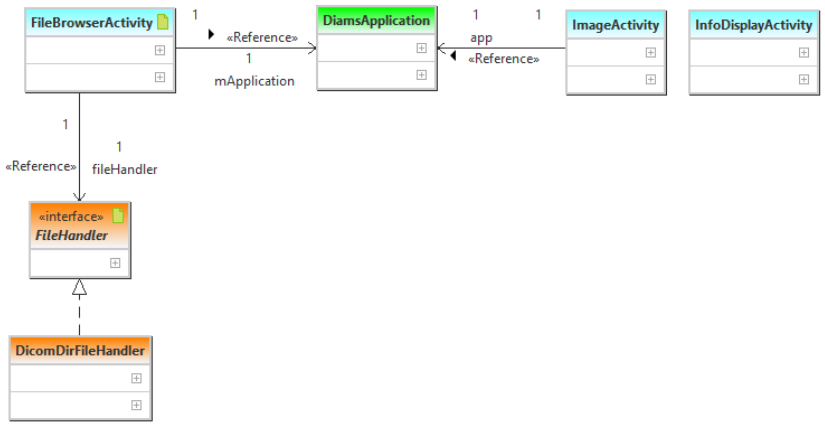

FIGURE 3.4 – Diagramme de classes de l'interface graphique de l'application

Chapitre 4

Travail

Sommaire

4.1 Travail 8
4.1.1 L'explorateur de fichiers 8
4.1.2 L'interaction avec un examen 8
4.2 Tests 8
4.2.1 Test 1 8

4.1 Travail

4.1.1 L'explorateur de fichiers

4.1.2 L'interaction avec un examen

4.2 Tests

Afin de garantir la fiabilité de nos livrable, nous réalisons une série de tests unitaires et de tests fonctionnels. Les tests unitaires nous permettent de nous assurer du bon fonctionnement de certaines parties déterminées du logiciel.

4.2.1 Test 1

Nous réalisons un ensemble de tests de blablabla. afin de blabla.

Chapitre 5

Bilan

Sommaire

5.1 Bilan **9**
5.1.1 Échecs 9
5.2 Si c'était à refaire... . . . . . **9**

5.1 Bilan

En conclusion, nous pouvons dire que le bilan général est plutôt bon. Nous avons globalement satisfait les besoins client tout en réalisant nos objectifs initiaux en terme de blablabla.

Au niveau de l'architecture, les méthodes sont restées simples, ce qui limite selon nous les sources de bugs. Nous limitons les dépendances avec les bibliothèques Java. L'application est donc plus maintenable et évolutive.

5.1.1 Échecs

Malgré ce bilan positif, nous déplorons blablabla.

5.2 Si c'était à refaire...

Si c'était à refaire, nous aborderions le problème différemment. En incluant nos connaissances en gestion de projets, nous pourrions aborder les problèmes liés à la complexification des phases d'intégration. En connaissant les patterns aux sein d'une équipe, nous aurions pu mesurer les effets de leur utilisation dans le cadre d'un projet utilisant les méthodes agiles, comme l'*XP* par exemple.

Table des figures

3.1 Diagramme de cas d'utilisation 4
3.2 Diagramme en couches de l'application 6
3.3 Diagramme de classes du modèle de l'application 7
3.4 Diagramme de classes de l'interface graphique de l'application . . 7

Rapport de projet de conduite de projet.
DIAMS or Neko tablet.
Master 2, 2012

Les sources de ce rapport sont disponibles librement sur https://github.com/mr-nours/rapport-cp.