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 l'application des méthodes agiles *Scrum* autour de la réalisation d'un logiciel d'interaction avec des fichiers médicaux au format *DICOM* sous tablette tactile *android*. Il s'agissait de développer des fonctionnalités dans un cadre simulant une situation professionnelle, en interaction avec un client et des équipes, tout en intégrant les bonnes pratiques acquises au cours de la formation académique.

Mots-clés : conduite de projet, agile, scrum, relations, communication

Table des matières

Chapitre 1

Gestion de projet

Sommaire

1.1 La Gestion du projet

1.1.1 Méthodes et outils Scrum

Le projet s'organise autour de la méthode agile *Scrum*, méthode qui nous a permis de maîtriser notre production, de la quantifier et de la planifier. Ce projet a été réalisé en groupe avec une durée fixe à respecter et des objectifs clairs exprimés en début de projet (confère cahier des charges).

Afin de piloter au mieux le projet, nous avons utilisé l'outil *Youkan*, qui permet de gérer tout le processus projet. Cet outil permet d'avoir une bonne traçabilité favorisant ainsi la collaboration entre les différents acteurs du projet. Nous avons utilisé *Youkan* pour planifier et suivre les différentes étapes de notre projet et modéliser les exigences du client en intégrant toutes demandes de modifications.

Nous avons employé la méthode *Scrum* pour établir les estimations sur la difficulté et la durée des différentes tâches à réaliser. Pour cela, nous avons mis en place un *backlog*, où nous avons listé les différentes tâches que nous avions à effectuer, et pour chacune d'entre elles, nous avons tous pondéré la tâche en fonction de sa difficulté à l'aide d'un système de points et de moyennes.

1.1.2 Découpage du projet

Le projet a été découpé en 3 sprints, un sprint équivalent à une période de 21 jours. C'est donc un processus itératif qui a été utilisé tout au long de la réalisation du projet. Le premier sprint consistait en l'évaluation des valeurs réelles de nos pondérations, c'est à dire à combien de jours réels correspondait une pondération. Ainsi, nous avons pu obtenir des estimations précises du temps nécessaire pour réaliser une tâche en fonction des compétences et de la cadence de travail de chacun des membres du groupe. De plus, *Youkan* dispose d'un Task Board faisant office de tableau virtuel sur lequel nous pouvons coller des post-its, post-it que nous pouvons classer dans une des quatre rubriques suivantes : Todo, In progress, Test et Done. L'ensemble des acteurs du projet peut se servir de ce tableau pour suivre avec précision l'avancée du projet.

1.1.3 Organisation de l'équipe

Afin de piloter l'équipe, un responsable de gestion nommé *Scrum Master* a été désigné. Son rôle est de s'assurer du bon déroulement du projet ainsi que de l'application de la méthode *Scrum*. Il a donc ainsi veillé à bien communiquer la vision et les objectifs du projet entre les acteurs de celui-ci. C'est lui qui était chargé d'échanger avec le client et de restituer, lors de réunions organisées, les différentes attentes du client ainsi que les différents objectifs à atteindre durant le sprint. Pour faciliter les échanges, le Scrum Master a fixé un jour dans la semaine afin de pouvoir échanger et faire le point avec l'équipe de manière hebdomadaire. Le Scrum Master a également accompagné l'équipe en demandant un retour des différents développeurs lors des entrevues quotidiennes, en répartissant le travail selon les compétences et préférences de chacun et en envoyant des mails de rappel ou de relance lorsque cela semblait nécessaire. Le Scrum Master a veillé au maintien d'une bonne cohésion de groupe, un point important du projet.

Enfin, nous avons été tout au long du projet critiques vis-à-vis de l'application de notre méthode de travail en essayant d'évaluer rétrospectivement et de manière objective, lors d'une réunion dédiée, les processus du projet (ce que nous avons du mal à appliquer, ce que nous devons améliorer, une estimation mauvaise à revoir, etc.).

1.1.4 Gestion des risques

A l'aide de *Youkan* et de la méthode *Scrum*, nous avons pu identifier et suivre nos risques plus efficacement. Nous avons pris l'habitude de mettre en place des solutions de contournements rapides afin d'anticiper les facteurs à risque pouvant ralentir la progression du projet tout en restant flexibles. Les paramètres de planifications ont également été surveillés de prêt grâce à la Burndown chart, qui représente graphiquement la charge de travail restante prévisionnelle et la charge de travail restante concrète.

1.1.5 Mise en place des jalons

Afin de garantir un projet fonctionnel, nous avons placé des jalons à chaque fin de sprint dans le but de fournir un livrable. Cette exigence nous a obligés a garantir un fonctionnement du produit tout au long du projet en déployant le produit réalisé sur tablette avant chaque livraison. Cela a également permis au client de voir l'avancée du projet et de pouvoir émettre immédiatement des remarques, des demandes ou des critiques concernant les livrables, nous permettant ainsi d'appliquer rapidement les correctifs nécessaires et d'aboutir à un produit fini correspondant au mieux aux attentes du client.

1.2 Environnement de développement

1.2.1 IDE

Eclipse est la référence pour le développement sur *android*, le système d'exploitation visé. *Eclipse* est multi-plateforme et résout le problème de l'utilisation de différents systèmes d'exploitation par les membres de l'équipe de développement. Son système de plugins permet de gérer le déploiement de tout le kit de développement *android*. Nous utilisons également le DDMS (Dalvik Debug Monitor Server), qui permet de détecter et de résoudre des bogues tout en interagissant avec le système.

1.2.2 Style de codage

Une convention de codage a été mise en place pour assurer une plus grande cohérence au sein de l'équipe. Cette convention est partie intégrante de la documentation développeur.

1.2.3 Matériel de déploiement

Nous utilisons une tablette tactile pour le déploiement de notre application. Cette tablette possède la version *android* 2.1 (API 7), qui n'est pas la plus récente. Par conséquent, certains services récents ne sont pas accessibles.

1.3 Outils de versionnage et techniques d'intégration

Le code est versionné sous git, pour que chacun dispose de son propre serveur. Cela procure l'avantage de réduire les **risques de conflits**. Le serveur web que nous avons choisi d'utiliser est GitHub. Il nous propose un traqueur de bogues, un système de commentaires du code pour un meilleur suivi du projet et un outil de visualisation de l'activité du projet sous forme de graphes d'arborescences. En sur-couche graphique de git, nous utilisons le logiciel *Smartgit*. Il propose des outils de comparaison de code et des journaux visuels. Nous versionnons le code en utilisant la méthode **PULL then PUSH** : seul le **Scrum Master** possède le droit d'intégrer des sources sur la branche master (branche principale). Les autres membres envoient leurs versions sur d'autres branches, mais seul le scrum master peut réaliser des fusions entre d'autres branches et

la branche master. Les autres développeurs peuvent demander une intégration de leurs modifications sur la branche principale par des *pull requests*. Nous utilisons ce système de branches pour séparer ce qui est en développement de ce qui est en phase d'intégration. L'arborescence représente aussi la répartition des fonctionnalités. Nous intégrons le travail régulièrement et par petites itérations pour éviter les pertes de temps liées à l'introduction de conflits.

1.4 Serveurs d'automatisations

1.4.1 Maven : déploiement et dépendances

Maven est un outil puissant permettant d'automatiser la mise en place d'un projet Java. Il gère les dépendances et les librairies utilisées. Cependant suite à des problèmes d'incompatibilité entre la version d'*android* utilisée et le système de templates de Maven, nous avons dû renoncer à l'utiliser. L'alternative utilisée est l'outil *Ant* et le générateur d'APK de Google (Application Package File) permettant d'installer des logiciels sous *android*. Les dépendances sont gérées manuellement, ce qui ne pose pas de problème car leur nombre est faible.

1.4.2 Jenkins : tests et compilation

Jenkins est un outil permettant via un serveur de réaliser automatiquement des tests. A chaque archivage du code, Jenkins peut exécuter les tests pour lesquels il est configuré et génère un compte-rendu ainsi que des journaux détaillés. La prise en main de Jenkins a nécessité du temps et son exécution se fait en local. Nous avons considéré et abandonné l'alternative proposée au CREMI, où le serveur est configuré pour fonctionner sur Savane, qui n'utilise pas git mais svn, un outil de versionnage moins puissant.

1.5 Analyse de code

Le projet est en Java, par conséquent la mémoire est gérer par un garbage collector. Toutefois, étant donné que l'application doit fonctionner sur une tablette, il est nécessaire d'éviter de surcharger cette mémoire (ouvrir trop de fichiers en même temps par exemple). Les outils d'analyse de code aident à éviter les mauvaises pratiques qui génèrent des fuites mémoire.

1.5.1 Sonar

Sonar permet une analyse poussée du code (duplications de code, détection de bogues, couverture par des tests etc.) mais il est basé sur Maven. Par conséquent nous ne pouvons pas l'utiliser. Ainsi nous nous sommes orientés vers un autre outil.

1.5.2 Lint

Lint, spécifique à *android*, permet la détection de certaines erreurs de conception types et précise si des éléments sont obsolètes par rapport à l'API visé, le but étant d'améliorer la lisibilité, la sécurité, la performance du code etc.

Chapitre 2

Déroulement du projet

Sommaire

2.1 Déroulement du projet

Le projet a été rythmé par une succession de 3 Sprints avec chaque semaine des réunions de projet.

2.1.1 Description du déroulement projet

Chapitre 3

Cahier des charges

Sommaire

Comme nous venons de le voir, notre projet consiste à nous intéresser à la réalisation d'un logiciel d'interaction avec des examen médicaux au format *DICOM* sur tablette tactile *android*. Définissons pour ce programme les objectifs du point de vue non-fonctionnel. Les besoins fonctionnels sont présentés au niveau de la figure 4.1 page 9 et détaillés au chapitre 4.1 page 8.

3.1 Besoins non-fonctionnels

3.1.1 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 nouvelles fonctionnalités induit le besoin de refactoriser souvent le code afin de simplifier la maintenance 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.

3.1.2 Ergonomie

Le besoin d'ergonomie se fait ressentir car les utilisateurs visés ne sont pas spécialement adeptes des dernières technologies informatiques et ne disposent pas du temps nécessaire au suivi d'une formation préalable à l'utilisation de l'application. L'application doit donc être au possible intuitive et facile à utiliser, malgré les contraintes matérielles. Pour un terminal dont la zone d'affichage est réduite, nous visons donc à maximiser la taille des contrôles tout en minimisant

leur interférence sur les zones d'affichage. Cela traduit aussi le besoin de rendre rapide l'accès aux informations des examens.

3.1.3 Performance

L'utilisabilité de l'application passe par ses performances. La puissance de calcul étant limitée sur les terminaux portables, l'enjeu principal est donc de limiter l'usage des ressources tout en maximisant la fluidité de l'interface graphique.

Chapitre 4

Architecture

Sommaire

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.

4.1 Diagramme des cas d'utilisation

Nous avons réalisé un schéma des cas d'utilisations, visible en figure 4.1 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 4.1 – 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.

4.2 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 4.2. En bleu, les packages de transformation des données. En vert, les packages utilitaires. En rouge, le cœur de l'application, qui inclut 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.

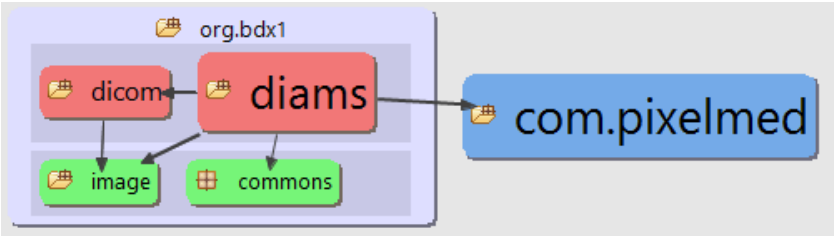

FIGURE 4.2 – Diagramme en couches de l'application

4.2.1 Le modèle d'examen

Voyons maintenant le diagramme des classes du modèle, en figure 4.3 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.

4.2.2 L'interface graphique

Nous nous intéressons maintenant au diagramme des classes simplifié de la partie graphique, visible en figure 4.4 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

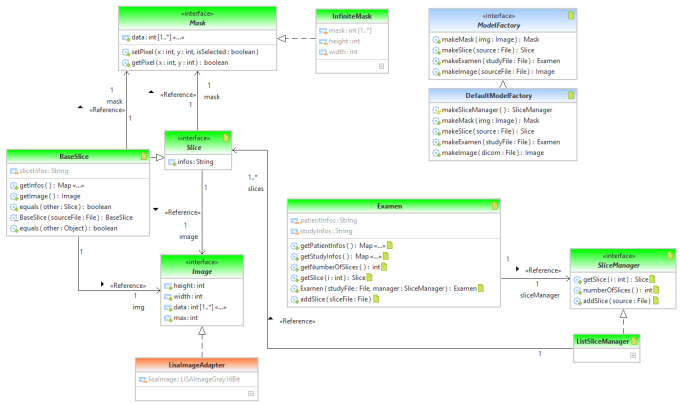

FIGURE 4.3 – Diagramme de classes du modèle de l'application

`ImageActivity` et `InfoDisplayActivity` permettent de visualiser les composants de l'examen choisi et d'interagir avec. La classe `DicomDirFileHandler` 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.

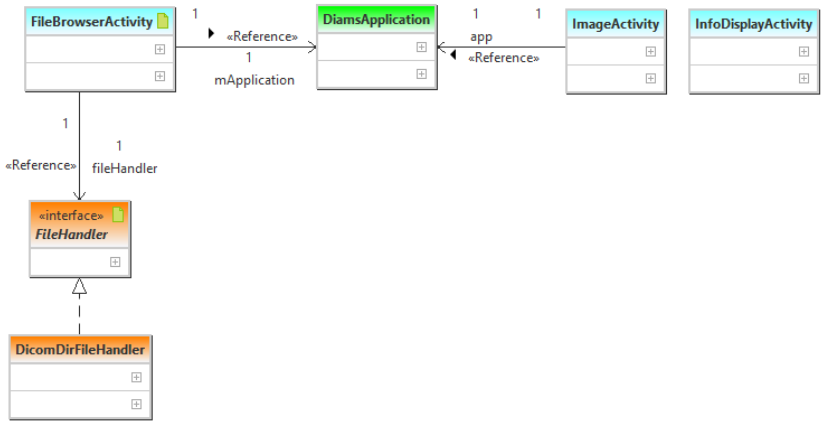

FIGURE 4.4 – Diagramme de classes de l'interface graphique de l'application

4.2.3 Le système de cache

Le développement d'une application mobile entraine des contraintes dues au matériel. La puissance du processeur et la quantité de mémoire vive sont limitées. Notre application devant gérer un grand nombre d'images, nous avons implémenté un système de cache afin de limiter l'occupation mémoire à un

instant t, sans pour autant entraver les actions de l'utilisateur par de longs temps de chargement.

Afin que notre couche modèle puisse être réutilisée dans un contexte n'entrainant pas l'utilisation d'un cache, nous avons fait attention à séparer les classes responsables de ce cache des autres classes modèles. Cette problématique nous a tout d'abord conduits à séparer la gestion des coupes de l'Examen proprement dit à travers l'interface `SliceManager`. Nous avons ensuite pu réaliser l'implémentation du cache indépendamment du modèle. La bibliothèque *android* contenant déjà un système de cache, nous avons décidé de l'utiliser pour implémenter `SliceManager`. De cette façon, une seule classe de notre système de cache dépend d'*android*. Enfin, nous utilisons une nouvelle `ModelFactory` héritant de `DefaultModelFactory` afin de faire la liaison entre le modèle et le cache.

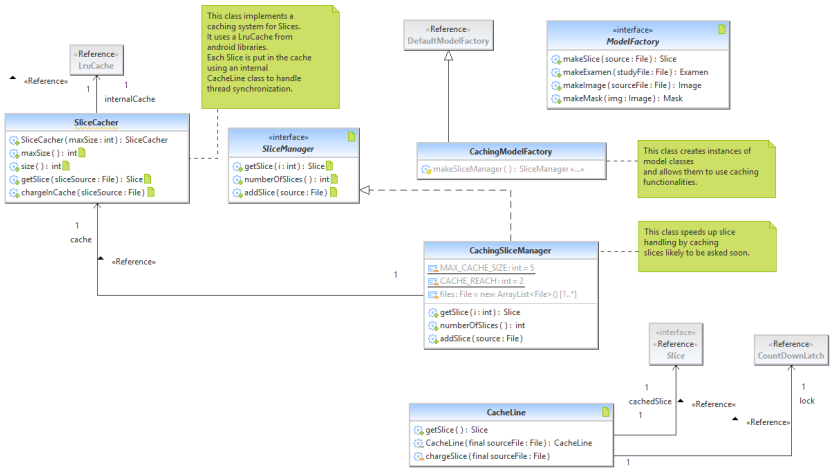

FIGURE 4.5 – Diagramme de classes du système de cache

Chapitre 5

Travail

Sommaire

5.1 Travail

Nous présentons l'explorateur de fichiers et l'interface d'interaction avec un examen, qui représentent les deux modules caractérisants la partie utilisateur de l'application.

5.1.1 L'explorateur de fichiers

L'ouverture de l'application conduit sur l'écran d'exploration des fichiers, visible en figure 5.1 page suivante. Nous distinguons les dossiers normaux des dossiers *DICOM*. Sur la figure 5.1 page suivante, on peut voir qu'un dossier a été entouré en rouge : il s'agit d'un exemple de dossier *DICOM*. Ces derniers représentent un examen et contiennent les fichiers qui constituent les données de l'examen. Le nom du dossier *DICOM* est masqué à l'utilisateur, qui voit à la place le nom, le prénom, le genre et l'age du patient associé à l'examen. Un clic sur un dossier quelconque place la vue à l'intérieur du dossier. On peut revenir dans le dossier parent par autre clic. Un clic sur un dossier *DICOM* déclenche l'ouverture de l'examen et l'utilisateur passe alors sur l'affichage présenté dans le chapitre suivant.

5.1.2 L'interaction avec un examen

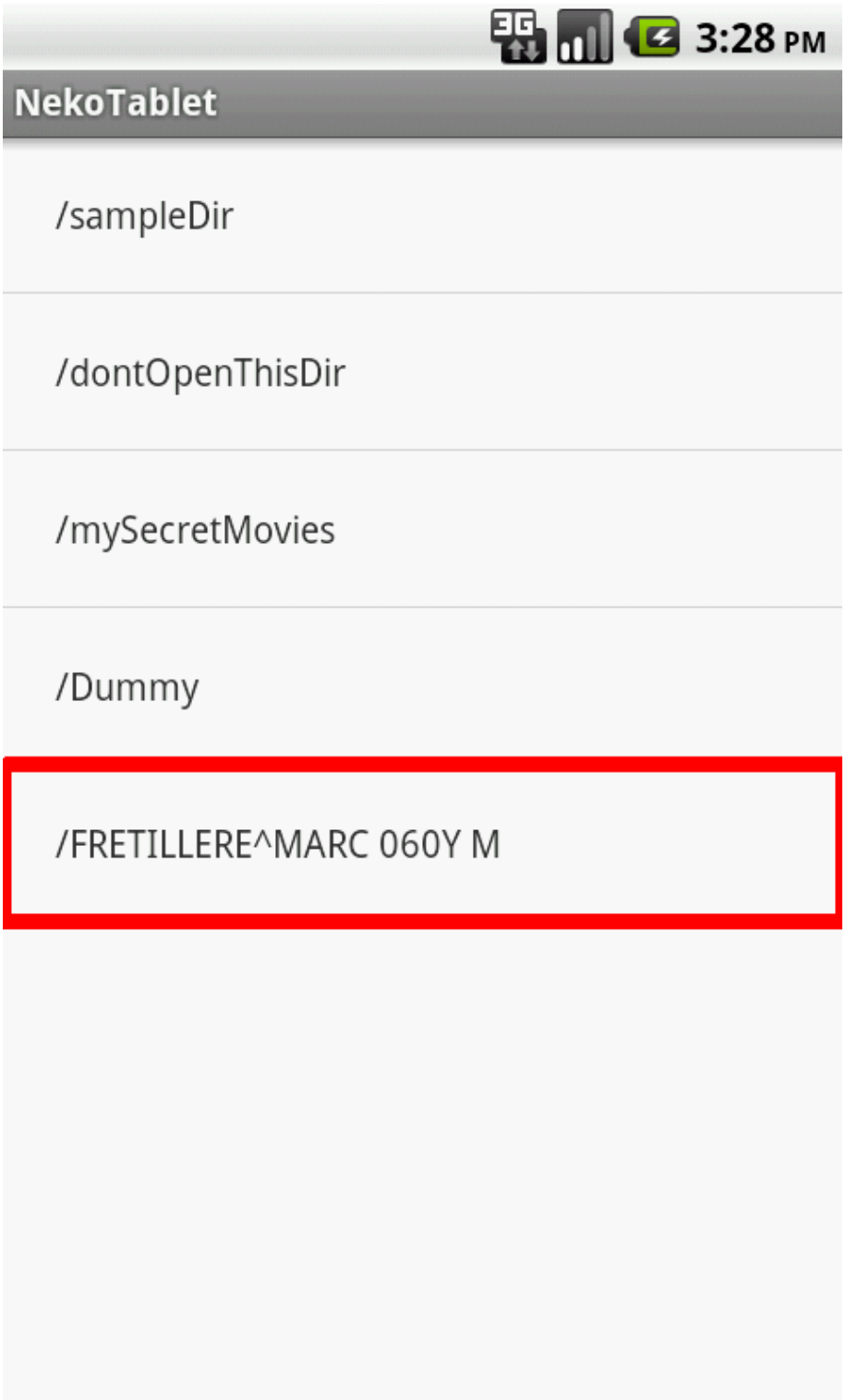

FIGURE 5.1 – Capture d'écran de l'explorateur de fichiers

Chapitre 6

Bilan

Sommaire

6.1 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.

6.1.1 Échecs

Malgré ce bilan positif, nous déplorons blablabla.

6.2 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

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.