Université Bordeaux 1
Master informatique
2011-2012

Rapport de Conduite de projet

Image2D Dicom

Alan Charpentier, Mickael Dales, Nicolas Moreaud, Abdallah Ndoye

Chargé de TD et Client : Hugo Balacey

Table des matières

Chapitre 1

Introduction

Ce projet s'inscrit dans le cadre de l'unité d'enseignement "Conduite de Projets". L'objectif est de nous familiariser avec les outils et les méthodes liés au développement logiciel. Ce projet constitue une simulation de conditions de développement réelles pour mettre en place et tester des pratiques de travail en équipe. Le projet est en relation avec le domaine de l'imagerie médicale, plus particulièrement avec celui de la radiographie. Notre travail consiste dans un premier temps à mettre en place une bibliothèque pour la gestion des examens radiographiques. Puis, dans un second temps, l'objectif est de concevoir une architecture client/serveur pour des traitements d'image. Un utilisateur (médecin) doit pouvoir à partir de son ordinateur sélectionner des examens et des traitements à effectuer dessus. Le serveur est alors chargé d'appliquer l'ensemble des traitements et doit fournir un résultat au client.

Chapitre 2

Gestion du projet

2.1 Environnement de développement

2.1.1 Environnement de développement intégré

Il est important pour l'équipe d'utiliser le même environnement de développement intégré. L'objectif est de pouvoir rapidement régler des problèmes liés à l'environnement de développement intégré et de toujours conserver la même configuration entre tous les membres de l'équipe.

Nous avons choisi QtCreator comme environnement de développement intégré. Celui-ci est adapté pour un développement avec du C++. Il offre une grande liberté de configuration. Nous avons notamment décidé des options de compilation que nous souhaitions utiliser.

Dans un souci d'uniformisation de notre développement et afin de rendre plus cohérent et lisible le code produit, nous nous sommes mis d'accord sur des règles de codage (convention pour les noms des identificateurs, des méthodes, etc...).

Nous nous sommes basés sur les fichiers project de QMake pour compiler le projet, car cette solution est très bien intégrée à QtCreator. Au cours du projet, nous avons pu évaluer les pours et les contres de ce choix.

Notre projet est constitué d'une librairie (le coeur, indépendant du besoin de l'application finale) et d'un exécutable qui utilise cette librairie. Nous gérons à la fois le code de la librairie et celui de l'application, nous avons modifié régulièrement les deux projets. Pour gérer ce cas, QMake autorise seulement de placer le projet librairie à l'intérieur du projet de l'application finale. Dans notre cas, cela n'est pas réellement une contrainte, car nous n'avons qu'un seule application finale. Mais dans un cadre de développement d'entreprise, cette solution n'est pas viable.

Nous avons tenté une approche différente pour gérer le problème : le projet de la librairie est placé à l'extérieur, dans son propre dossier. Il compile normalement, mais s'installe dans ses propres dossiers "bin" et "include". En bricolant un peu, on pourrait permettre à l'utilisateur de choisir le dossier d'installation (par défaut /usr/) et au développeur de pouvoir l'installer dans le dossier qui contient le code de la bibliothèque. Mais avec cette technique, on ne sépare pas les versions compilées en debug et en release, et on conserve d'anciens fichiers headers qui ne sont supprimés que lorsque la compilation de la bibliothèque fonctionne à nouveau, engendrant régulièrement des confusions. Par ailleurs, il semble que QMake soit un outil destiné à mourir à petit feu avec l'arrivée de son successeur QBS.

Ces raisons nous forcent à penser que CMake serait un meilleur choix pour les prochains

projets développés basés sur Qt, parce qu'il est moins limité que QMake et sait générer toutes sortes de fichiers projets propres aux différents IDE. Une des difficultés rencontrées était la configuration des ".pro", il s'agissait de définir les chemins pour les fichiers sources, des entêtes et des librairies utilisées. Qt étant bien documenté nous n'avons pas eu de mal à bien comprendre comment ça marche.

2.1.2 Gestionnaire de versions

Git a été choisi comme gestionnaire de versions pour notre projet. Son avantage vient de l'utilisation des branches, qui deviennent rapidement indispensables à mesure que le nombre de personnes travaillant sur le projet augmente. Nous avons donc mis en place un protocole de travail pour l'utilisation de Git. Premièrement, tous commits sur la branche Master devaient être réalisés par une seule et même personne (élue en début de projet). Ceci avait pour objectif de limiter les conflits entre le travail de chaque membre du groupe. Cette méthode a l'avantage de garantir une certaine sécurité dans les commits, cependant elle nous a posé quelques problèmes car nous devions attendre que la personne responsable de Git réalise les merges pour être certain que les changements apportés au code n'entraînent aucun conflit. Deuxièmement, nous avons décidé qu'il n'était pas possible d'effectuer un commit tant que le code ne compilait pas.

Nous avons choisi Bitbucket comme dépôt pour notre projet. Ce dernier a l'avantage d'être gratuit et privé. Le caractère privé de ce serveur d'hébergement était important car il nous avait été demandé de ne pas rendre disponible sur Internet notamment le code qui nous avait été fourni en début de projet. Nous avons bien sur donné accès au git à notre client pour qu'il puisse suivre les avancements de notre projet.

Afin d'avoir à notre disposition toute la puissance de Git et de faciliter notre développement, il nous fallait notamment un outil pour visualiser les commits effectués. Un tel outil a l'avantage de récapituler les changements effectués et la personne qui en est à l'origine. Cela permet en cas de problèmes de compréhension du code de savoir à quel membre du groupe s'adressait pour obtenir des explications. En fait, Bitbucket met à notre disposition un tel outil : visualisation de l'ensemble des changements effectués avec l'auteur, possibilité de comparer deux fichiers afin de faire ressortir les modifications effectuées entre deux versions.

Bibucket a aussi la bonne idée d'intégrer un bugtracker afin que nous puissions faire remonter à toute l'équipe les problèmes rencontrés. Au final, Bitbucket est un très bon choix car il centralise de nombreux services indispensables.

Nous avons eu des difficultés à utiliser git parce que nous connaissions mal ce gestionnaire de version. Nous avons souvent eu des blocages lors de l'utilisation des commandes commit (avec certaines modifications qui n'étaient pas répertoriées), push et pull (la version locale du projet ne correspondait pas toujours à la dernière version disponible). Au fur et à mesure, nous avons noté les commandes utiles et gagné en expérience.

Par exemple, certains fichier d'un projet ne doivent pas être versionnés. Il s'agit des fichiers binaires compilés et des fichiers de configuration locaux. Ces fichiers ont au début du projet gêné certains membres du groupes parce qu'il fallait systématiquement spécifier de ne pas versionner ces fichiers. Après quelque semaines, nous avons résolu le problème en ajoutant un fichier .gitignore à la racine du projet, qui permettait d'ignorer automatiquement ces fichiers.

Certains membres de l'équipe ont eu du mal à savoir ce que faisaient les autres programmeurs, ou à avoir une vision d'ensemble du projet. En effet, en travaillant presque indépendamment dans des branches séparées il fallait switcher de branche pour connaître les

modifications des autres personnes, or cette opération était parfois difficile. Il fallait d'abord avoir pushé ses modifications, et ne pas avoir de conflits sur sa branche. Dans le cas contraire, il fallait faire un "git checkout –" pour ne pas prendre en compte les modifications sur la branche courante.

2.2 Gestion de configuration

Bitbucket a facilité notre gestion de configuration car il nous a permis de centraliser toutes les informations pertinentes au bon déroulement de notre projet. La facilité d'accès de toutes ces informations nous a fait gagner du temps.

2.2.1 Wiki

Bitbucket offre la possibilité de mettre en place un wiki. Nous l'avons utilisé afin de centraliser des informations pratiques. A chaque fois qu'un membre de l'équipe rencontrait une difficulté et qu'il la résolvait, il inscrivait la solution trouvée sur le wiki. Les informations que nous avons fait figurer sur le wiki sont :

- La liste des commandes qui nous ont été utile pour gérer le dépôt git
- La mise en place de l'IDE pour compiler et modifier le projet
- La résolution des problèmes liés aux incohérences de l'IDE
- La liste des requêtes du serveur demandées par le client

Nous avons également pensé publier sur le wiki les points clés des réunions équipe-client et intra équipe pour en conserver une trace. Cela devait nous permettre de clarifier l'évolution du projet avec le client, en particulier les axes de développement, les délais à tenir et les fonctionnalités demandées afin de justifier les choix de l'équipe. Ce point n'a pas été mis en oeuvre concrètement car cela prenait trop de temps par rapport au temps passé à développer.

2.2.2 Référents

En plus du wiki, nous avons mis en place des référents. Un référent est une personne compétente dans un domaine et capable d'aider les autres membres de l'équipe pour des problèmes en relation avec son domaine. Par exemple, nous avions un référent pour Git. En effet, tous les membres de l'équipe n'étaient pas familiarisés avec Git au début du projet, il nous a donc paru intéressant que la personne maîtrisant déjà ce gestionnaire de versions puisse aider les autres membres de l'équipe. Pour ce faire, nous avons fait dès la première semaine une mini formation sur Git au cours de laquelle la personne qui avait déjà utilisé Git a pu faire profiter les autres membres de l'équipe de son expérience.

2.3 Méthode de développement - Scrum

La méthode de développement de notre projet nous a été imposée. Il a été décidé que nous devrions travailler avec la méthode agile Scrum. Cette dernière impose une certaine rigueur tout au long du cycle de développement d'un logiciel. Cette section a pour objectif de présenter comment nous avons travaillé avec Scrum. Notre premier acte en tant qu'équipe a été l'élection du Scrum master : personne qui garantit que l'équipe applique bien la méthode Scrum.

2.3.1 Backlog

Le backlog a été la première étape de notre projet. Il décrit les objectifs que nous avons à remplir pour arriver au bout du projet. L'écriture du backlog a été une étape compliquée car pour chacun des membres de l'équipe c'était notre premier contact avec une méthode agile plus encore avec Scrum. Nous n'avons pas complètement réussi (au moins dans un premier temps) à distinguer les tâches qui devaient être dans le backlog de celles qui ne devaient pas l'être. Nous pensions à tort au début du projet que le backlog ne pouvait contenir que des tâches en liaison directe avec le cahier des charges. Cette ignorance nous a posé des problèmes lors des premiers sprints : nous sélectionnions des tâches beaucoup trop complexes pour être réalisées en un seul sprint.

Nous avons eu du mal à respecter la méthodologie Scrum qui vise à établir les user stories avec le client puis à les prioriser. Au contraire, après nous avoir décrit le logiciel qu'il voulait obtenir, le client nous a demandé d'écrire nous même les user stories pour ensuite en vérifier avec nous le contenu. Cela a été un léger frein parce que les user stories ne sont pas exactement celles que le client aurait souhaitées la première fois, et parce que cela cache des détails qui auraient pu être explicités dès le début. Par exemple, nous n'avions pas compris que le format BMI3D n'était pas un format standardisé. Par la suite nous avons cru comprendre que nous étions libre du contenu des fichiers BMI3D (basés sur un parser pré-existant) alors que le client souhaitait en réalité que le format soit commun aux différents groupes de la matière. Cela a donc entraîné des surprises au cours de l'avancement du projet, et un manque de traçabilité et de précision qui ont nuit à la communication claire et objective avec le client.

Dès la première réunion, il a semblé très peu probable que nous ayons le temps de réaliser l'ensemble des user stories décrites par le client, c'est pourquoi nous avons essayé au fur et à mesure de négocier les tâches à réaliser et d'informer le client de nos possibilités. Nous avons cherché quel était aux yeux du client le but réel de ce projet universitaire. Bien sûr, il s'agissait de nous apprendre comment se gère un projet avec la méthodologie Scrum, et de prendre connaissance des outils nécessaires à un développement de qualité. Mais le projet avait également une teneur technique et une composante métier qui n'étaient pas à négliger : notre client n'avait pas choisi les sujets au hasard. Au fur et à mesure des semaines, nous avons compris que ce projet devait en partie servir au client à trouver de bonnes idées pour un projet de même teneur en matière d'architecture et de choix des technologies. Nous avons donc essayé dans la mesure du possible, quand nous devions faire un compromis ou abandonner une tâche, de favoriser les tâches les plus utiles au client et à notre formation.

2.3.2 Sprints

Nous avons réalisé au total trois sprints de trois semaines. Nous ne comptons pas là dedans la première semaine que nous avons consacré à la rédaction du backlog et la mise en place de l'environnement de développement : cette première semaine correspondait en fait au sprint 0. Un sprint débutait par le résumé de ce qui avait été fait au sprint précédent. Nous choisissions ensuite en équipe les tâches que nous souhaitions réaliser au cours des trois semaines à venir. Puis chacun choisissait la ou les tâches sur lesquelles il avait envie de travailler. L'objectif étant de ne pas forcer un membre de l'équipe à accomplir une tâche qui ne l'intéresse pas. Avec une équipe de quatre personnes, chaque tâche finissait par trouver preneur.

Le premier sprint a principalement consisté à compiler les sources existantes (coeur du programme) et à choisir les technologies utilisées.

Certaines classes étaient obsolètes et d'autres manquante. Il n'était pas évident de diviser le travail pour avancer ensemble sans modifier les mêmes choses ou créer de conflits car les classes dépendaient les unes des autres. Nous avons fini par travailler à deux sur les fichiers existants, et à deux pour écrire les classes manquantes.

Le second sprint avait pour but de pouvoir afficher des examens (luminosité, contraste), d'établir la liste des requêtes serveur et base de données nécessaires et de coder l'interaction avec la base de données. Nous nous sommes alors penché sur les technologies envisageable pour la création du serveur.

A ce moment là nous avions déjà bien avancé sur le coeur. Nous nous sommes adaptés à l'environnement de travail du Crémi ou une partie des librairies utilisées étaient manquantes. Nous avons bénéficié de l'aide de Mr Delmon, administrateur du crémi, qui a accepté d'installer toutes les librairies nécessaires (dont dcmtk) au bon déroulement du projet.

Nous avons un peu perdu le fil des sprints sur la fin. Nous avons du faire des choix avec le client parce qu'il n'était pas envisageable de traiter la totalité des user stories. Nous avons donc travaillé sur l'application des filtres, la finitions des parsers (format MI3D commun aux différentes équipes, parsing de l'ensemble des fichiers d'un dossier en un seul examen), et sur la mise en place de l'IHM.

Nous avons pris le temps de chercher au cours du projet quelle technologie devait être utilisée pour programmer le serveur, et nous avons testé ces technologies, mais nous n'avons pas eu le temps de programmer les requêtes.

2.3.3 Réunions

Chaque semaine nous avions au moins deux réunions : une avec le client et une avec l'équipe.

2.3.3.1 Daily scrum

Le Daily scrum avait lieu environ deux fois par semaines : une avant la réunion avec le client, et une autre en fin de semaine. Il était l'occasion de mettre à jour le backlog, de tenir chacun au courant des retours du client et de se concerter sur les éléments à faire remonter, sur les décisions à prendre, sur les objectifs à atteindre. Au début du projet cette réunion avançait au rythme des idées qui nous venait. En lisant un livre sur la méthode scrum, cette méthode à petit à petit gagné en organisation. Le Scrum Master animait cette réunion, en posant à chaque membre de l'équipe trois questions :

- Qu'as-tu terminé depuis la dernière réunion ?
- Quels sont les problèmes qui te gênent ?
- Que penses-tu pouvoir terminer d'ici la prochaine réunion ?

Ainsi, quand un problème survenait, nous pouvions soit réaffecter les tâches soit travailler à plusieurs sur une même tâche le temps de la résoudre. La méthode Scrum a permis de donner une dynamique au développement du projet. Personne n'est resté bloqué sur une tâche sans que les autres membres de l'équipe ne viennent l'aider. De fait, certaines semaines nous faisions plusieurs réunions, notamment pour débloquer des situations.

Ces réunions ont dans l'ensemble été un point positif pendant toute la durée du projet parce que nous trouvions facilement le temps de nous retrouver grâce à la disponibilité de l'ensemble des membres de l'équipe.

2.3.3.2 Réunion client

Les réunions hebdomadaires avec le client étaient l'occasion de présenter le travail réalisé au cours de la semaine et de poser des questions au client. Quand nous sentions que nous n'arriverions pas à achever entièrement le projet, ces réunions servaient également à prioriser les tâches restantes.

Le client a bien joué le rôle de product owner, car il sait précisément quel résultat il souhaite obtenir, et car il a une bonne connaissance des habitudes et des attentes des utilisateurs du logiciel à développer. En sa qualité de client technique, il a un avis sur les technologies utilisées et l'architecture mise en place.

2.3.4 Youkan

Youkan est l'outil que nous avons utilisé comme support à la méthode Scrum. Son utilisation a été pour nous un casse-tête et un échec. Nous n'arrivions pas à définir quelle était la bonne granularité pour l'écriture des tâches d'un sprint. Nous nous sommes trompés de sens pour l'évolution des tâches : nous pensions que le nombre de point collectés était croissant, alors qu'il s'agissait plutôt d'un compte à rebours. Nous avons inversé la tendance à partir du milieu du second sprint, mais le burndown chart est resté inutilisable. Ceci a été problématique dans le sens où nous n'avons jamais pu utiliser Scrum (via Youkan) pour planifier la fin de notre projet par exemple.

D'autre part, Youkan en tant que tel nous a posé des problèmes : il est aussi peu intuitif qu'ergonomique. Tout cela réunit a fait que nous n'avons pas pu profiter au maximum des possibilités offertes par la méthode Scrum (prévision de la fin du projet notamment). Néanmoins, Youkan nous a permis d'avoir graphiquement une vue des tâches à accomplir au cours du sprint en cours.

2.4 Environnement d'intégration continue

Ne disposant pas d'un serveur, nous avons cherché un service web où installer gratuitement un serveur Jenkins. Le service Google App Engine, dont l'usage est gratuit pour de petites applications, donne la possibilité d'installer des archives WAR. Cependant il ne satisfait pas les conditions nécessaires pour faire tourner Jenkins. En effet, l'écriture de fichier n'est pas autorisée sur les serveurs de Google.

Nous avons trouvé le service CloudBees qui propose également des solutions gratuites de cloud computing. En particulier, CloudBees donne l'accès à ses utilisateurs à un dépôt git ou svn, à un serveur Jenkins et à des serveurs de déploiement. Cependant, l'installation de plugins est limitée à une courte liste de plugins prédéterminée.

Nous n'avons donc pas trouvé de solution gratuite pour faire fonctionner Jenkins. Nous avons demandé à Etienne Baudimont (un étudiant de notre promotion) qui dispose d'un hébergement gratuit de nous permettre d'utiliser son serveur Jenkins, mais le serveur était instable.

Christophe Delmon, administrateur du Crémi, n'était pas d'accord pour installer Jenkins sur notre demande, il voulait être en contact direct avec l'enseignant qui demandait d'installer Jenkins.

Après négociations, le chargé de TD a accepté de demander à Christophe Delmon d'installer Jenkins avec l'aide de Nicolas qui avait déjà configuré Jenkins à plusieurs reprises.

La contrainte d'être hébergé par les serveurs de la fac est que le dépôt git devait être hébergé dans l'espace de stockage de la fac.

Après avoir fait un tour rapide des logiciels de gestion d'un dépôt Git (gitk, gitg, git-cola, etc.), l'équipe a choisi de ne pas utiliser Jenkins, et de conserver le dépôt et l'interface très pratique de BitBucket.

Configuration :

Pour utiliser git, depuis Jenkins, il faut donner au service qui gère le dépôt git la clé publique SSH du serveur Jenkins. Il faut au préalable la générer dans le dossier /var/lib/jenkins/.ssh/ car Jenkins ne la génère pas par défaut.

Le serveur Jenkins travaille dans la branche "jenkins" du dépôt Git, c'est là que le scrum master fait les push.

Notre projet utilise Qt, et notamment les QWidgets. Lorsque nous lançons l'application en mode test unitaires sur le serveur d'intégration continue, l'affichage des widgets est désactivé dans le code (à l'aide de macros) car ils ne sont pas nécessaires. Cependant, la simple instanciation d'un QWidget produit des communications avec le serveur graphique. Afin de pouvoir tester les méthodes des classes héritant de QWidget, nous a choisi d'utiliser le pluggin XVFB de Jenkins qui simule l'exécution d'un serveur X11.

Nous utilisons le pluggin xUnit pour recevoir les résultats des tests unitaires.

Si la compilation du projet fonctionne et que tous les tests unitaires passent avec succès, le serveur Jenkins fait un push sur la branche Master. Cette branche représente la branche stable du projet. Si la compilation échoue, le serveur Jenkins envoie un mail aux développeurs. Cela arrive également si les tests ne passent pas, la version est alors marquée instable dans Jenkins.

Bien que cette configuration aie été faite, elle n'a pas été utilisée en pratique.

Chapitre 3

Architecture

3.1 Utilisation de l'application

L'application que nous développons n'est pas terminée. En l'absence d'une partie de ses composants, les différentes fonctionnalités peuvent être testées par le biais d'un programme en ligne de commande. Pour cela, nous instancions les fenêtres, les filtres et les parsers dans la classe Main. L'usage de l'application est le suivant :

- usage : path_to_a_dicom_directory -options1 -option2...

En outre nous définissons les options :

- -h : pour obtenir de l'aide
- -gui : pour afficher l'interface graphique
- -gaussian : pour spécifier qu'il y a le filtre Gaussien à appliquer
- -sbmi path.bmi3d : pour exporter le fichier sous le format bmi3d
- -snmi path.nmi3d : pour exporter le fichier sous le format nmi3d
- -onmi path.nmi3d : pour ouvrir un fichier au format nmi3d
- -v : pour afficher les informations concernant le patient
- -login : pour lancer l'authentification

3.2 Choix des technologies

3.2.1 Serveur

Nous sommes à l'origine partis sur la base d'une architecture REST, conformément à la demande du client. Cependant, C++ n'est pas un langage très adapté pour produire une architecture REST. Contrairement à Java qui dispose d'annotations simples pour exposer des fonctions sur le web, il existe peu de librairies orientées WebServices pour C++. Pour REST, on citera les librairies : Apache Axis C, C2Serve, WSO2.

Le framework Apache Axis C++ est basée sur Apache Axis C et ne propose qu'un sous ensemble des fonctionnalités du framework C. C2Serve est une petite librairie, agréable d'utilisation, mais qui ne gère pas la sérialisation des données, et qui est peu documentée. Enfin, WSO2 est un serveur d'application basé sur Apache Axis C.

Rajoutons un critère de sélection : les transferts de données entre client et serveur doit pouvoir se faire sur des quantités de données importantes (fichiers de 500 Mo).

Afin d'éviter les ralentissements, on souhaite soit gérer la socket à la main (indépendamment des autres requêtes), soit utiliser un mode qui garantisse des temps de transfert courts.

C2Serve ne permet pas de gérer cela. Apache Axis permet d'utiliser MTOM (Message Transmission Optimization Mechanism), mais le projet d'exemples livré avec le code source produit une erreur de segmentation pour des fichiers de plus de 20 Mo.

Nous avons donc abandonné l'approche REST qui aurait demandé de gérer tout à la main à partir d'une simple socket.

En élargissant le champ de recherche, nous avons décidé d'utiliser le framework gSoap, permettant de produire un webservice service SOAP et utilisant MTOM avec succès (exemple des sources mtom-stream). gSoap sérialise et désérialise les données automatiquement à la volée, permet par défaut l'authentification du client et la compression gzip des données transférées via la zLib.

Le code d'envoi et de réception de fichier a été trouvé parmi les exemples de gSoap fournis avec les sources. Il fait 600 lignes, il est assez imbuvable, mais doit se retrouver dans un fichier à part dont on ne modifiera que de très petites parties. Les performances sont très largement supérieures à l'envoi d'un fichier découpé sans MTOM : en local, si l'on envoie un fichier en utilisant la requêtes "sendFile(string name, vector<char>, int offset, int size)", on ne parvient qu'à transférer 3 Mo à la seconde. En utilisant MTOM, on passe à plus de 80Mo à la seconde.

3.2.2 Base de Données

Nous avons choisit une base de donnée SQLite qui permet d'intégrer directement la base de données au serveur. Cela enlève donc la relation client serveur des SGBD habituels. Cette base de donnée est extrêmement légère, ce qui nous permet de garder la mémoire du serveur pour les fichiers.

Nous avons opté pour le module QDatabase de Qt pour implémenter la base de données. Grâce à l'utilisation de cette librairie, nous pouvons changer de base de données en modifiant seulement deux paramètres à l'ouverture. On pourra donc migrer vers une base de données MySQL ou Oracle facilement.

3.2.3 Moteur d'application de filtres

Nous avons mis beaucoup de temps avant de nous pencher sur l'architecture des filtres. Le code fourni par le client contenait une grosse boite de dialogue permettant de choisir une liste de filtres à appliquer sur l'image, avec une prévisualisation du résultat. Une partie du code était manquante et a été rajouté pour que la boite de dialogue puisse fonctionner à l'exécution. Cette étape a nécessité du temps car elle a révélé plusieurs bugs dans le coeur du projet.

Cette boite de dialogue était essentielle dans le projet : elle contenait tout le mécanisme existant pour appliquer des filtres, chaque filtre étant implémenté sous la forme d'une simple fonction prenant quelque paramètres flottants en entrée, et une image et un masque en entrée/sortie. La client souhaitait rendre cette partie du programme plus extensible (cette classe était assez complexe à modifier de par sa taille) en extrayant le mécanisme d'application des filtres dans une classe à part et en le généralisant. Il voulait pouvoir écrire ses filtres sous la forme de pluggins compilés séparément. De plus, nous devions permettre au programme de demander l'application d'un filtre par le réseau, les valeurs des arguments devait donc être sérialisable.

Les objectifs étaient donc de :

- Supprimer le switch qui liste les filtres existants afin de synthétiser le code et de permettre l'ajout dynamique de filtres sur le serveur sans modification du code client
- Décrire les arguments (nom, type, valeurs possibles) pour que l'utilisateur entre des valeurs cohérentes
- Gérer des filtres dont les prototypes diffèrent
- Pouvoir générer l'IHM qui permet de lancer le filtre
- Décrire chaque filtre dans une classe afin de pouvoir l'instancier quand il est chargé comme un plugin

La solution retenue est la suivante :

- Un argument de filtre est représenté par un objet qui décrit son nom, son type, sa valeur courante et optionnellement sa valeur par défaut, sa valeur minimum, sa valeur maximum et son domaine de définition (énumération).
- On utilise une classe de type variant (Boost.Variant, QVariant...) pour encapsuler le type concret et la valeur de l'argument. Cette classe gère automatiquement la sérialisation des données pour pouvoir les transférer sur le réseau ou les sauvegarder dans un fichier. Dans la pratique on a utilisé la classe QVariant, extensible avec de nouveaux types de données, mais on aurait également pu mettre à profit la classe DataSet de Tulip qui donne une interface simplifiée et permet donc de refactoriser plus facilement le programme en cas de besoin.
- Un filtre est constitué d'un nom et d'un ensemble d'arguments pouvant être parcouru facilement. On peut accéder à un argument à partir de son nom, ce qui est fortement utile quand on connaît l'algorithme que l'on souhaite appliquer.
- Deux classes héritent de filtre : Filtre2D et Filtre3D qui diffèrent sur les types de données sur lesquelles les filtres peuvent s'appliquer.
- Une classe FilterWindow permet d'afficher une boite de dialogue Qt où l'utilisateur peut rentrer ses valeurs. Cette boite de dialogue est générée à la volée en prenant en compte les types de données attendues et les contraintes sur les valeurs.

Cette architecture répond parfaitement aux besoins cités ci-dessus, elle augmente en revanche le couplage entre ce projet à la librairie Qt. On notera que la description des arguments et des filtres est sérialisable, que les données des arguments sont également sérialisable, mais qu'on ne peut pas ajouter de contraintes particulières ou sérialiser l'implémentation des filtres. Par exemple, on ne peut pas obliger un argument à n'accepter que des valeurs impaires, ou à ne pas prendre les valeurs -4, 8 et 19.23. Il faudrait un langage de description du domaine de définition pour parvenir à ce résultat. Donc, avec architecture, on est obligé de demander au serveur si les arguments sont conformes aux contraintes de l'implémentation concrète du filtre.

3.2.4 Outil de tests

Nous avons utilisé le framework CppUnit pour écrire les tests unitaires. Pour pouvoir utiliser conjointement les assertions, nous avons redéfini les assertions pour qu'elles lancent une exception plutôt que d'arrêter brutalement l'exécution du programme l'on est en "mode test".

Chapitre 4

Implémentation

4.1 Refactoring

Le refactoring a constitué une partie importante du temps passé à développer. Tout d'abord parce que le développement du projet a commencé par trois semaines où principal était de parvenir à faire compiler le projet et à obtenir une exécution cohérent. Ce n'était pas gagné d'avance : plusieurs classes étaient manquantes, et on ne parvenait pas toujours à déterminer devait être le résultat de certaines méthodes. Nous citerons par exemple une classe d'image que nous n'avions pas avec méthode getPointWidth() qui renvoyait en réalité la largeur d'une image qu'elle contenait. Heureusement, nous avons bénéficié de l'aide de notre client lors d'échanges réguliers d'emails.

Nous avons apporté plusieurs modifications significatives à l'architecture du programme. Pour commencer, nous avons créé deux classes différentes pour les images 2D et les images 3D afin de mieux les distinguer. Nous avons essayé de standardiser les identifiants utilisés. En effet, le code contenait beaucoup de variables mask et volume, mais les données contenues variaient d'une instance à l'autre. Dans notre projet, nous avons déclaré le type Mask qui est un typedef sur une image 3D de booléen et le type Volume : typedef sur une image 3D de valeurs flottantes (niveaux de gris).

De façon générale, nous avons essayer d'augmenter le niveau d'encapsulation des données dans tout le programme, et en particulier dans les classes d'examen et d'images. Ce travail est laborieux car il entraîne des modifications dans tout le programme et force à créer de nouvelles classes ou à modifier le fonctionnement là où le premier code accédait à des variables publiques. Le point le plus ennuyeux a été d'encapsuler les données de l'examen : à force d'avoir des bugs qui émergeaient d'une initialisation partielle de l'examen, nous avons décidé de supprimer les setters les plus gênants et de forcer la création d'une objet cohérent. Cependant, le code des parsers fonctionnait par petite touche en remplissant petit à petit les variables publiques ou en utilisant les setters de la classe Examen ce qui a rendu le changement difficile. De plus, nous avons compris sur la fin que les fichiers Dicoms devaient souvent être chargés ensemble pour former un examen complet. Dans ce cadre, il serait utile d'avoir un builder ou une classe chargée de l'instanciation de l'examen, qui garantirait que l'examen instancié est complet et permettrait d'enlever une plus grande partie des setters de la classe Examen.

Enfin, le code source contenait beaucoup de références à des classes Manager, qui agissaient comme des annuaires d'objets. Ces managers servaient à définir un point d'accès global dans le programme à tel ou tel objet. Nous n'avions en revanche pas le code source de ces classes

Manager et ne savions pas précisément comment et quand les objets étaient instanciés. Après plusieurs concertation, nous avons choisi de supprimer ces classes et de faire passer les objets à l'origine référencés par les managers dans les constructeurs des objets qui les utilisaient. Si nous voulions que le le serveur fonctionne bien, et que le coeur s'adapte à tout type d'application, il fallait éviter d'avoir des variables globales uniques. Si ces modifications sont vraiment nécessaires pour le fonctionnement du serveur, elles ont demandé beaucoup trop de travail par rapport au gain réellement apporté.

4.2 Parser Bmi3D

Concernant le parser Bmi3D, nous nous sommes mis d'accord sur un format standard avec les autres groupes. Notre choix s'est voulu le plus générique possible. Seul le stockage des informations posaient des problèmes. Ainsi, la procédure pour le stockage des informations correspond à écrire le nombre des informations à stocker, puis pour chaque information, stocker la taille son intitulé, son intitulé, la taille de sa description et sa description. Ce schéma de stockage permet une lecture et une écriture très génériques.

4.3 Niveaux de gris

Les fichiers Dicom contiennent une ou plusieurs images dont les valeurs sont mesurées selon l'échelle de Hounsfield. Cette échelle permet de déterminer quel est le type de matériau représenté par les pixels de l'image. Pour des raisons d'encodage et de précision des valeurs, chaque constructeur d'appareil de mesure radio choisit sa propre échelle de valeurs, et fournit deux valeurs slope et intercept qui permettent de faire la conversion. Dans le programme que nous avons reçu, les images sont chargées en RAM telles qu'elles sont stockées dans le fichier DICOM. Dans plusieurs parties du programme, il était difficile de savoir si une donnée, par exemple le niveau de gris minimum de la fenêtre de visualisation, était exprimée en Hounsfield ou en unité arbitraire. Parfois nous convertissions plusieurs fois une valeur en HounsField sans nous en rendre compte. C'est pourquoi nous avons mis en place une classe pour représenter un niveau de gris avec plusieurs méthodes pour récupérer cette valeur dans les différentes unités en présence.

Dans divers endroits du programme on utilisait des valeurs de contraste/luminosité, dans d'autres endroits on se servait des seuils minimum et maximum de la fenêtre de visualisation. Nous avons généralisé cela dans une classe FenetreNiveauDeGris qui gère d'une part le contraste et la luminosité de l'affichage, et d'autre part les seuils minimum et le maximum, ces valeurs étant intrinsèquement liées.

4.4 Interface Homme Machine

Nous avons placé les classes relatives aux fenêtres de l'application dans un dossier gui. Nous nous sommes aidés de QtDesigner (section Design de QtCreator) pour réaliser les interfaces graphiques plus facilement. Pour chaque fenêtre nous avons donc un triplet (.h .cpp .ui)

Le dossier gui est composé trois classes ; la classe SliceViewer qui gère l'affichage de l'image, les deux classes LoginFrame et NewAccount qui gèrent l'authentification.

- La classe SliceViewer permet d'afficher les coupes de l'examen radiologique à partir de l'examen. Elle délègue l'affichage concret à la classe ImageViewer qui ne sait afficher

que des images simples. Ensuite pour régler le contraste et la luminosité de l'image nous utilisons les méthodes de la classe GrayViewWindow.cpp. Cette classe nous permet en effet de régler le Contraste, la Luminosité mais aussi le min et le max. Nous connectons leurs valeurs aux valeurs deux QSliders de notre fenêtre, ainsi nous réglons en faisant appel à notre méthode update() à chaque fois que les valeurs des sliders changent. Nous avons aussi connecté les valeurs des QSlider à deux QLineEdit qui affichent les valeurs du contraste et de la luminosité actuelles. Enfin, pour visualiser différent types de tissus du corps humain (os, abdomen, tissus mou...), nous avons prédéfini des valeurs de contraste et de luminosité. Pour le zoom nous avions envisagé quelques solutions. Elle n'ont pas aboutit, nous sommes par conséquent restés avec le zoom de base déjà implémenté. Une des difficulté que l'on a rencontré sur les fenêtres était la possibilité de régler le contraste et la luminosité directement en cliquant gauche et en déplaçant la souris, nous n'avons pas eu le temps d'implémenter ce mécanisme. La solution à laquelle nous avions pensé était de définir des QPoint qui mesuraient la distance entre le point où l'on clique et le point où l'on relâche la souris. Avec la projection de cette droite sur les axes verticaux et horizontaux nous aurions été capable que régler le contraste et la luminosité.

- La classe LoginFrame est la première fenêtre instanciée au lancement de l'application. Elle contient deux champs à remplir "Login" et "Password" et trois boutons "Créer Compte", "Annuler" et "Valider". En appuyant sur le bouton "Valider" nous instancions la fenêtre principale (SliceViewer). La base de donnée n'a pas été intégrée avec l'IHM. L'application accepte donc pour le moment n'importe quelles valeurs en entrée. Le bouton "Créer Compte" quand à lui instancie la fenêtre "New Account" qui permet de créer un nouveau compte.

4.5 Filtres

Seul un filtre a été implémenté pour montrer la validité de l'architecture. Il s'agit du filtre gaussien 2D, prenant un paramètre flottant. Le code de description de l'interface est très simple, et l'IHM générée est tout à fait satisfaisante. Pour pouvoir étendre ce code, il faudra :

- Placer le code de chaque type de données supportées dans une classe à part. Par exemple, la valeur d'un argument de type flottant doit être rentré par l'utilisateur par le biais d'un QDoubleSpinbox
- Ajouter la vérification des contraintes de valeurs des arguments dans la méthode canApply de Filter

On pourra également mettre en place un numéro de version des filtres pour la compatibilité, et une priorité d'exécution, comme par exemple : pré-filtre = 5, filtre = 10, post-filtre = 15. De cette manière, on peut rajouter des filtres "intermédiaires" dans le pipeline.

4.6 Base de données

Les requêtes sur la base de données ont été implémentés dans une même classe. Cette classe permet à l'utilisateur de :

- ouvrir ou la base de données (lors de l'ouverture si la base n'existe pas, on la crée automatiquement avec toutes ces tables
- créer et supprimer un docteur (l'utilisateur)

- connecter un docteur
- insérer et supprimer un examen
- changer le nom d'un examen
- lister les examens d'un docteur
- connaître l'espace disque disponible pour un docteur

4.7 Serveur

Nous n'avons pas eu le temps d'implémenter le serveur.

4.8 Tests Unitaires

Les tests unitaires couvrent les parties suivantes du coeur du programme :
- Niveaux de gris
- Fenêtre de visualisation (contraste/luminosité)
- Parser MI3D
- Base de données

Chapitre 5

Conclusion

Ce projet a été l'occasion pour chacun des membres de l'équipe de s'initier à la méthode agile Scrum. Nous avons également pu nous rendre compte d'à quel point il était important de disposer d'un bon environnement de développement commun à toute l'équipe. Ce projet était le premier pour lequel l'accent était mis avant tout sur la gestion du projet. Nous avons pu voir quelles étaient les difficultés rencontrées dans la mise en place d'une réelle gestion de projet et bien entendu quels étaient les bénéfices en retour. La méthode Scrum a plu à l'ensemble de l'équipe. Elle donne une vraie dynamique au développement du projet tout en mettant l'accent sur le travail d'équipe.